An Introduction to Nintendo DS Graphics

Version 2.4.0

The content of this document is highly confidential and should be handled accordingly.

Table of Contents

TWL-06-0156-001-A
Released: July 8, 2009

List of Tables

List of Figures

Revision History

Version	Revision Date	Description
2.4.0	2009/03/10	• Changed the title to "An Introduction to Nintendo DS Graphics" to correspond with the added support for TWL. • Changed hardware structure and related content to add TWL information. • All references to NITRO in headings and in the text were either changed to Nintendo DS or deleted. • Updated the content of section 7.3 Using Graphics Display Mode—Example 3 (Displaying 3D on Two Screens) to that of the latest sample demo.
2.3.0	2004/09/13	• Added a cautionary note to section 4.2.7 Supplemental 1: Deciding a Polygon's Color
2.2.0	2004/06/21	• Corrected the description of extended palette • Corrected the specular component formula for light calculations • Added supplement to the antialiasing description
2.1.0	2004/03/23	• Added section 2.3 Display System Block Diagram • Added Chapter 7 Appendix
2.0.0	2004/01/28	Initial version.

1 Introduction

This manual was designed to provide an introduction to the graphics features of the Nintendo DS (TWL and NITRO). Therefore, the following issues are not addressed in detail for actual in-game use of each feature:

- How much ROM and RAM is consumed
- How much processing time is required by each processor
- The ease of use or ease of applicability of each feature
- Actual programming code

This manual was written based on the contents of *TWL Programming Manual*, Version 1.4.

2 Overview of DS Graphics Hardware

2.1 Hardware Structure

This section describes in detail the important hardware structures for DS graphics.

2.1.1 CPU Core

The processor is made up of two CPU cores, called ARM9 and ARM7:

- ARM9 Clock Frequency 67.028 MHz (TWL Double Speed Mode; 134.056 MHz)
- ARM7 Clock Frequency 33.514 MHz

2.1.2 Graphics Engine

The DS contains graphics engines for the control of both 2D and 3D graphics. The TWL graphics engine provides control over whether to enable use of a circuit that corrects bugs in NITRO. Otherwise, there are no significant changes to the specifications.

- 2D Graphics Engine A
- 2D Graphics Engine B

...Controls the 2D Graphics Engine
(Clock Frequency: 33.514 MHz)

- 3D Graphics Engine
  - GE (Geometry Engine) ...Controls calculation processes for 3D Graphics
(Clock Frequency: 33.514 MHz)
  - RE (Rendering Engine) ...Controls rendering processes for 3D Graphics
(Clock Frequency: 33.514 MHz)

Note: 2D Graphics Engine B has limited graphics features.

2.1.3 Memory

There are three categories of memory that can be used. Refer to Table 2-1 Memory and Capacities

for memory details and capacities.

- Main Memory
- VRAM (9 Banks: VRAM-A through VRAM-I)
- Work RAM (2 types for NITRO Compatible Mode: ARM9/ARM7 Shared Memory and ARM7 Dedicated Memory, 4 types for TWL Mode: ARM9/ARM7 Shared Memory, ARM7 Dedicated Memory, ARM9/ARM7/DSP Instruction RAM Shared Memory, and ARM9/ARM7/DSP Data RAM Shared Memory)

Table 2-1 Memory and Capacities

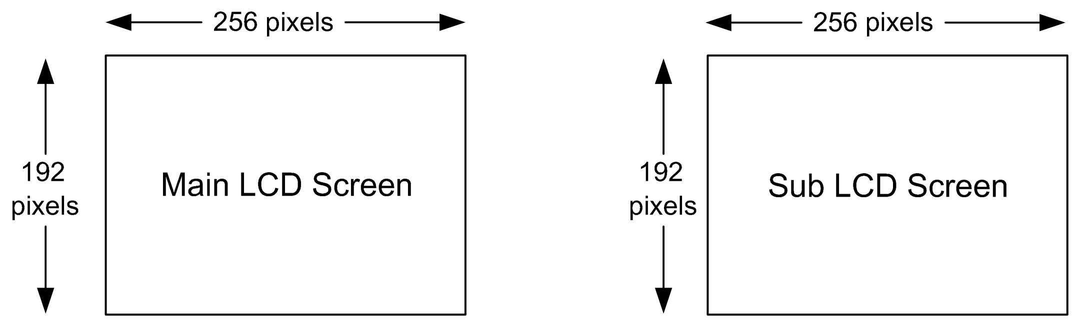
<table>
<tr><th colspan="2">Memory</th><th colspan="2">Capacity</th></tr>
<tr><td colspan="2">Main Memory</td><td colspan="2">16 Megabytes
(4 Megabytes in DS Compatible Mode)</td></tr>
<tr><td rowspan="9">VRAM</td><td>VRAM-A</td><td>128 kilobytes</td><td rowspan="4">Total
512 kilobytes</td></tr>
<tr><td>VRAM-B</td><td>128 kilobytes</td></tr>
<tr><td>VRAM-C</td><td>128 kilobytes</td></tr>
<tr><td>VRAM-D</td><td>128 kilobytes</td></tr>
<tr><td>VRAM-E</td><td colspan="2">64 kilobytes</td></tr>
<tr><td>VRAM-F</td><td>16 kilobytes</td><td rowspan="2">Total
32 kilobytes</td></tr>
<tr><td>VRAM-G</td><td>16 kilobytes</td></tr>
<tr><td>VRAM-H</td><td>32 kilobytes</td><td rowspan="2">Total
48 kilobytes</td></tr>
<tr><td>VRAM-I</td><td>16 kilobytes</td></tr>
<tr><td rowspan="2">Work RAM
(DS compatible)</td><td>ARM9 / ARM7 Shared</td><td colspan="2">32 kilobytes</td></tr>
<tr><td>ARM7 Dedicated</td><td colspan="2">64 kilobytes</td></tr>
<tr><td rowspan="4">Work RAM
(TWL)</td><td>ARM9 / ARM7 Shared</td><td colspan="2">32 + 256 kilobytes
(Total 288 kilobytes)</td></tr>
<tr><td>ARM7 Dedicated</td><td colspan="2">64 kilobytes</td></tr>
<tr><td>ARM9 / ARM7 / DSP Instruction RAM Shared</td><td colspan="2">256 kilobytes</td></tr>
<tr><td>ARM9 / ARM7 / DSP Data RAM Shared</td><td colspan="2">256 kilobytes</td></tr>
</table>

Note: DS main memory corresponds to the GBA's CPU External Work RAM. DS Work RAM corresponds to the GBA's CPU Internal Work RAM.

2.1.4 LCD

There are two LCDs: a main LCD and a sub LCD. The size of each LCD is 256 × 192 pixels, as shown in **Error! Reference source not found.**.

Figure 2-1: LCD Display Size



2.2 Display Mode

Nintendo DS has three display modes for the display of images on its LCD screen:

- Graphics Display Mode
- VRAM Display Mode
- Main Memory Display Mode

2.2.1 Graphics Display Mode

This is the mode that displays graphics generated by the 2D and 3D graphics features.

2.2.2 VRAM Display Mode

This is the mode that displays image data prepared in VRAM. VRAM-A through VRAM-D can be used for this display.

2.2.3 Main Memory Display Mode

This is the mode that displays image data stored in main memory.

2.3 Display System Block Diagram

The numbers in Figure 2-2, below, correspond to the following explanations:

1. 2D Graphics Engine A

   Generates OBJ and BG graphics. See Chapter 3 2D Graphics for details.

2. 2D Graphics Engine B

   Generates OBJ and BG graphics. See Chapter 3 2D Graphics for details.

3. 3D Graphics Engine

   Generates 3D graphics. See Chapter 4 3D Graphics for details.

4. OBJ

   See section 3.2 OBJ Features for details

5. BG

   See section 3.3 BG Features for details.

6. Layering and Color Special Effects

   See section 3.6 Color Special Effects and Chapter 5 Merging 2D and 3D Layers for details.

7. VRAM

   This is memory used for storing image data for character, screen, palette, texture, and bitmap sources. VRAM is referenced as needed from the 2D Graphics Engines and the 3D Graphics Engine at the time of rendering. In VRAM Display Mode, the bitmap data in VRAM can be displayed.

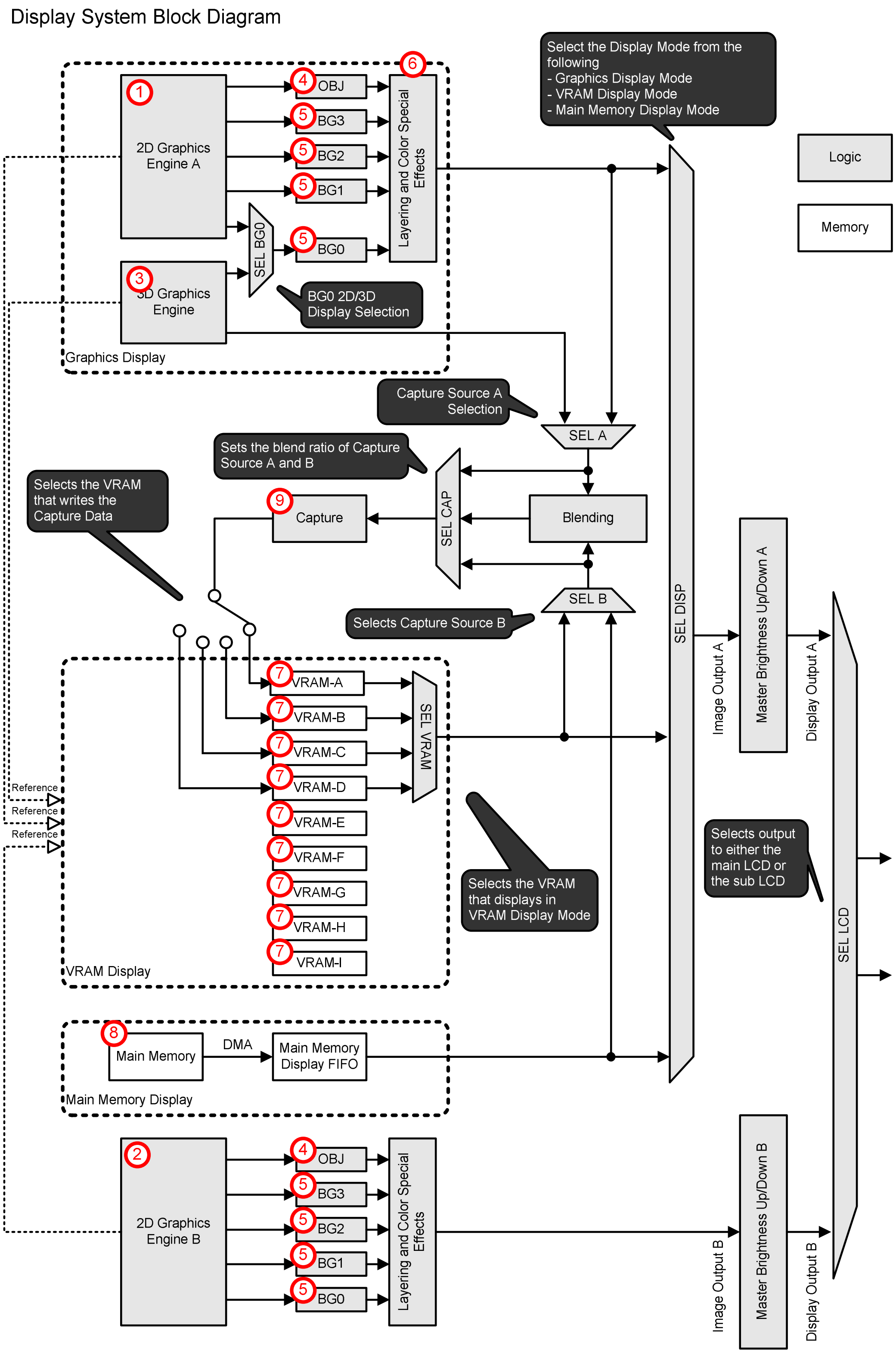
8. Main Memory

   In Main Memory Display Mode, the bitmap data in Main Memory can be displayed.

9. Capture Feature

   This is a feature that takes images generated outside of 2D Graphics Engine B into VRAM-A through VRAM-D. This is explained in Chapter 6 Display Capture.

Figure 2-2 Display System Block Diagram



3 2D Graphics

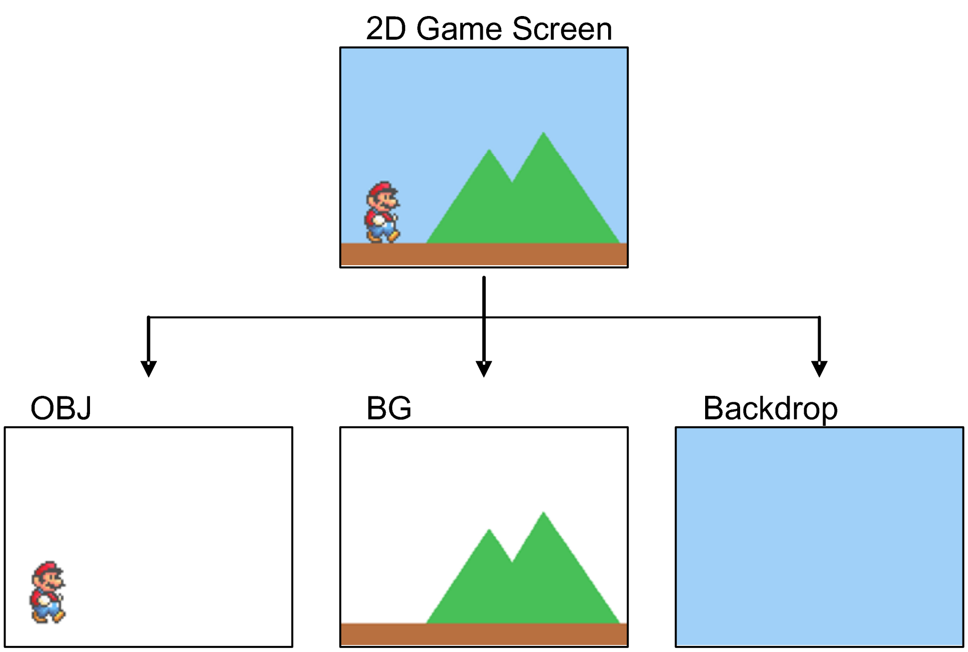
Nintendo DS, for the most part, conforms with the GBA's 2D graphics display functionality.

3.1 2D Screen Composition

The 2D game screen is composed of three main parts. (See Figure 3-1.)

- OBJ (Object)
- BG (Background)
- Backdrop

Figure 3-1 Structure of the 2D Game Screen



3.1.1 OBJ (Object)

OBJs are generally displayed objects that move around onscreen (such as the player, enemies, or items). OBJs are displayed according to the following sequence. (See Figure 3-2.)

1. Prepare the character data on which each OBJ is based.

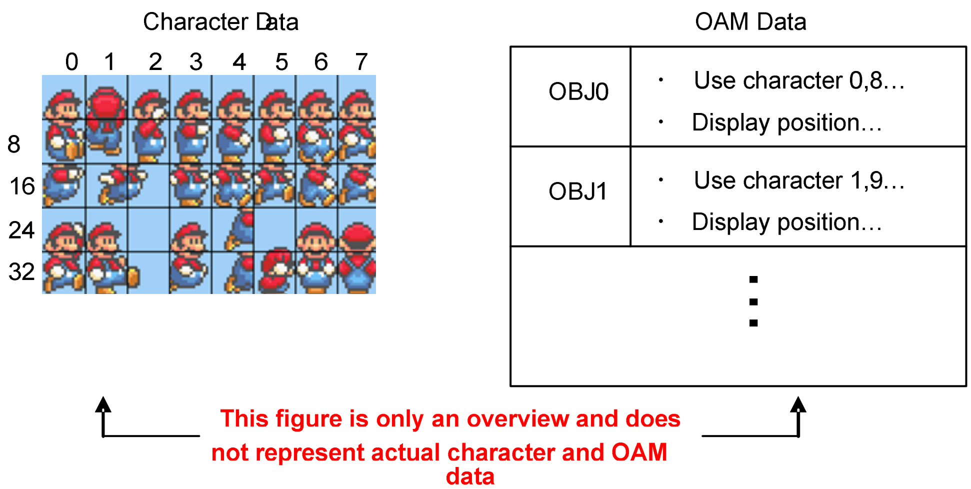
⇩

2. Prepare data for each OBJ, for all ways that the character will be used in the character data, what position it will be displayed in, etc. This data is known as "OAM (Object Attribute Memory) Data."

⇩

3. Render and display the OBJ.

Figure 3-2 Overview of Data Needed for OBJ Display



Nintendo DS can display a maximum of 128 OBJs in one rendered frame.

For OBJ features, refer to section 3.2 OBJ Features.

3.1.2 BG (Background)

BG is used to display the "background" of a game. There are two methods for displaying BG, outlined below.

3.1.2.1 Character Method BG (See Figure 3-3)

1. Prepare the character data used as a base.

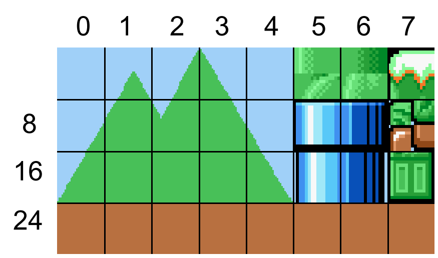
   ⇩

2. Prepare the data that indicates which numbered character and in what position it will be placed. This is known as the "screen data."

   ⇩

3. Render and display the BG.

Figure 3-3 Overview of Data Needed for Character Method Background Display



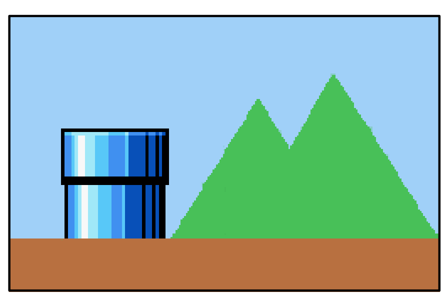
Screen Data

0	0	0	0	0	0	0	0
0	0	0	0	1	2	3	0
0	13	14	8	9	10	11	12
0	21	22	16	17	18	19	20
24	24	24	24	24	24	24	24

This figure is only an overview and does not represent actual screen data and character data

⇩

Resulting BG Screen

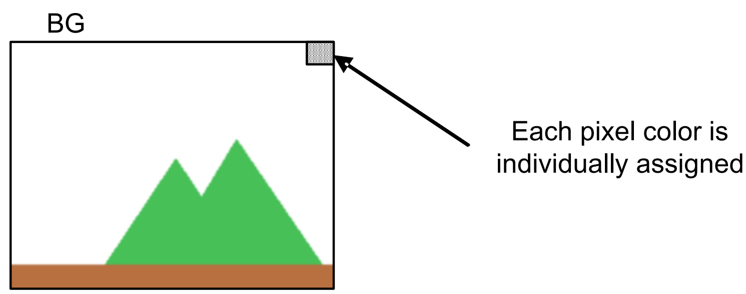


3.1.2.2 Bitmap Method BG

This is a method of rendering and displaying the BG where each pixel's color is directly assigned without the use of either character or screen data.

There are two ways to designate color: use of the color palette, or direct assignment of RGB values. (See Figure 3-4.)

Figure 3-4 Bitmap Method BG



For details about BG features, see section 3.3 BG Features.

3.1.3 Backdrop

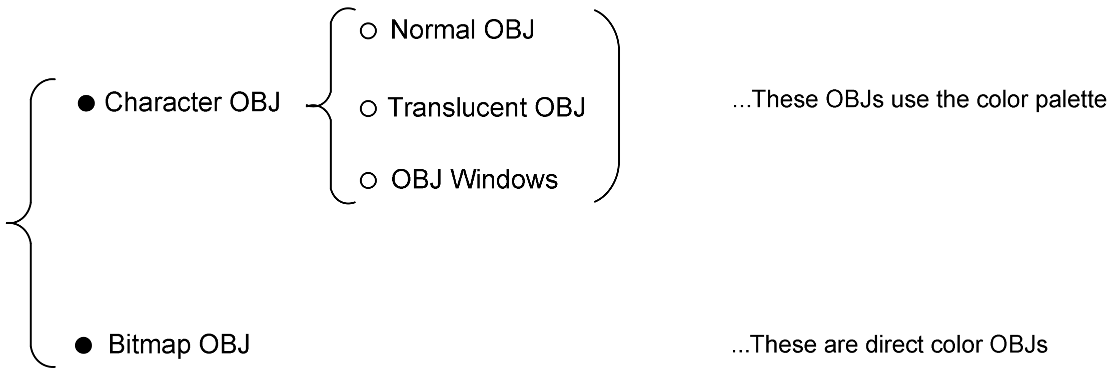
The backdrop is the color displayed in the region that is not obstructed by either the onscreen OBJ or BG.

The backdrop is composed of color data only.

3.2 OBJ Features

There are two kinds of OBJ: character OBJs and bitmap OBJs.

There are three kinds of character OBJ: normal OBJs, translucent OBJs, and OBJ windows.



Caution: Character OBJs and bitmap OBJs are character formats, since both use character data. Bitmap OBJs are not bitmap methods.

Table 3-1 indicates which settings and features can be applied to each OBJ.

Table 3-1 OBJ Features

<table>
<tr><th rowspan="2">Feature</th><th colspan="3">Character OBJ</th><th rowspan="2">Bitmap OBJ</th></tr>
<tr><th>Normal OBJ</th><th>Translucent OBJ</th><th>OBJ Window</th></tr>
<tr><td>Number of
Colors / Palettes</td><td colspan="3">1) When using the standard palette
16 colors × 16 palettes
256 colors × 1 palette
2) When using the extended palette
16 colors × 16 palettes (standard palette)
256 colors × 16 palettes (extended palette)</td><td>32,768 colors</td></tr>
<tr><td>Number of Characters</td><td colspan="3">1024 – 8192 characters (in 16-Color Mode)
512 – 4096 characters (in 256-Color Mode)</td><td>256 characters, or 1024 – 2048 characters</td></tr>
<tr><td>OBJ Size</td><td colspan="4">8×8 16×8 32×8 32×64
8×16 16×16 32×16 64×32
8×32 16×32 32×32 64×64</td></tr>
<tr><td>Number of OBJs Displayable on One Screen</td><td colspan="4">128 OBJs (Converted to 64 × 64 pixels)</td></tr>
<tr><td>Number of OBJs Displayable on One Line</td><td colspan="4">128 OBJs (Converted to 8 × 8 pixels)</td></tr>
<tr><td>Translucency Processing</td><td>△</td><td>○</td><td>△</td><td>△</td></tr>
<tr><td>Window Features</td><td>×</td><td>×</td><td>○</td><td>×</td></tr>
<tr><td>Affine Conversion</td><td>○</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Double-Size Display</td><td>○</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>HV Flip</td><td>○</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Mosaic</td><td>○</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Priority</td><td>○</td><td>○</td><td>×</td><td>○</td></tr>
</table>

3.2.1 Number of Colors / Palettes

This is the number of colors and palettes that can be used by an OBJ. Bitmap OBJs do not use the color palette, so they represent the number of colors displayed. For a more detailed description of color palettes, see section 3.4 Color Palette.

3.2.2 Number of Characters

This is the number of characters that can be defined within character data when the basic character is an 8×8-pixel character. Depending on how the character data is positioned in VRAM, the number of definable characters will change.

3.2.3 OBJ Size

Twelve sizes of OBJ can be selected, ranging from 8×8 to 64×64.

3.2.4 Maximum Number of OBJs Displayable on One Screen

This is the maximum number of OBJs that can be displayed on one screen.

3.2.5 Maximum Number of OBJs Displayable on One Line

This is the maximum number of OBJs that can be displayed on one line.

3.2.6 Translucency Processing

This indicates whether α-blending has been applied with color special effects processing.

The △ mark on the table above means that the certain restrictions will be applied when enabling α-blending to the OBJ.

For a more detailed description of color special effects processing, see section 3.6 Color Special Effects.

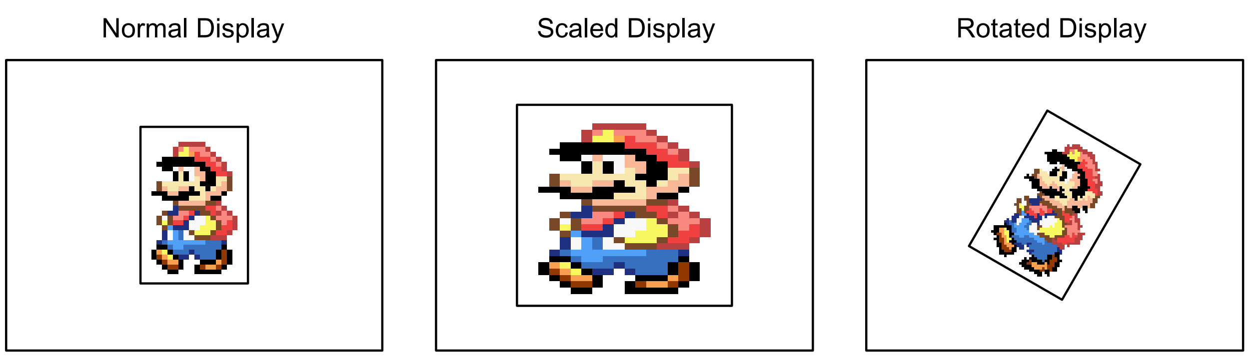
3.2.7 Window Features

For a more detailed description of window features, see section 3.5 Windows.

3.2.8 Affine Conversion

This feature deals with the rotated and scaled display of an OBJ. (See Figure 3-5.)

Figure 3-5 OBJ Affine Conversions

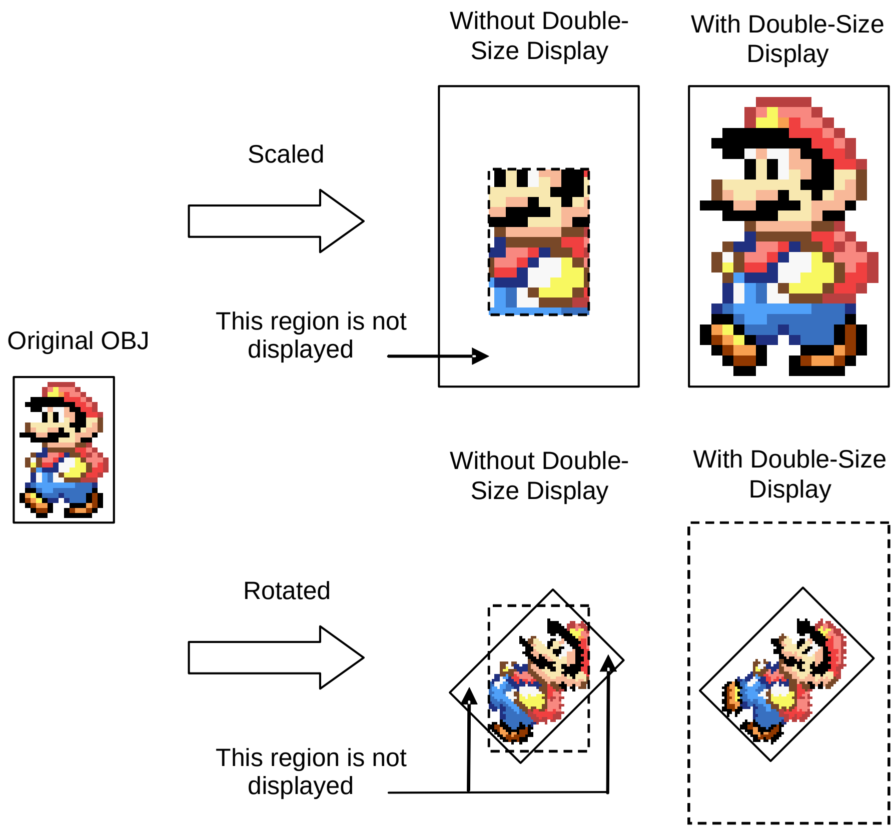


Note: An OBJ that uses rotation and scaling features is known as an Affine OBJ.

3.2.9 Double-Size Display

When using affine conversion features, the region of the OBJ that exceeds the OBJ's normal size is not displayed. When using double-size display, the entire region up to where the OBJ doubles in size will be displayed, as shown in Figure 3-6.

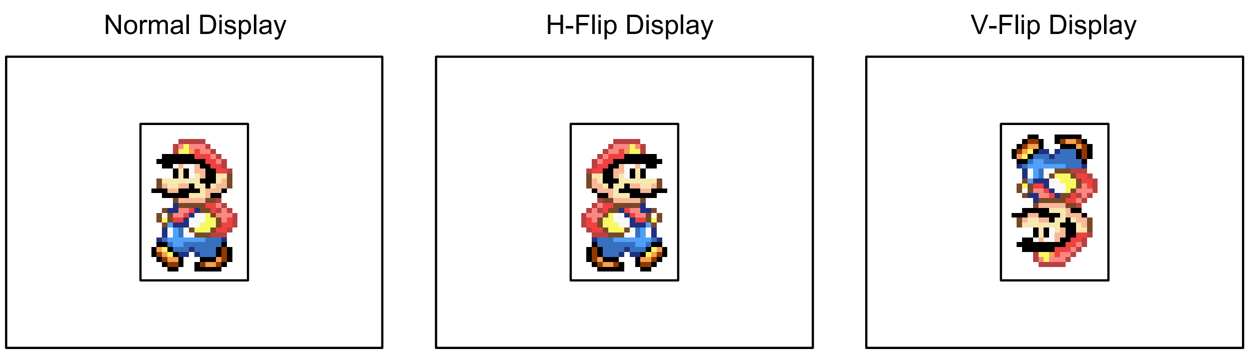
Figure 3-6 OBJ Double-Size Display



3.2.10 HV Flip

This feature displays reversed horizontal and vertical OBJs. (See Figure 3-7.)

Figure 3-7 OBJ H/V Flip



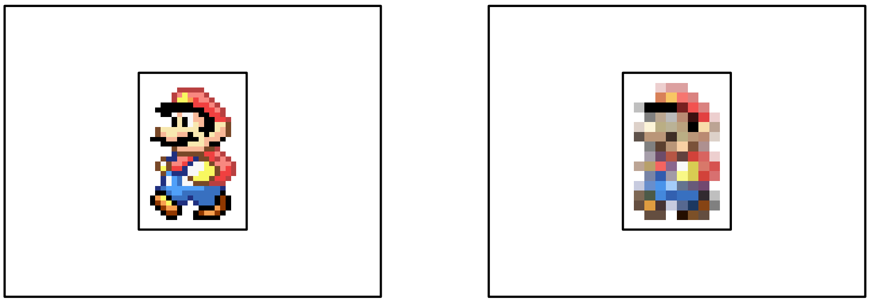
3.2.11 Mosaic

This feature displays mosaics.

The number of pixels in a mosaic can be specified. (See Figure 3-8.)

Figure 3-8 OBJ Mosaic

Normal Display   Mosaic Display



3.2.12 Priority

There are four levels of display priority. The OBJ with the highest priority is displayed in front. Priority cannot be assigned to the OBJ window.

3.3 BG Features

There are two methods for BG: character method and bitmap method.

- Character Method ··· This method uses screen data to refer to character data.
- Bitmap Method ··· This method directly specifies the color value and color index value without using screen data and character data.

There are five different types of BG, divided over the character method and the bitmap method. These BG types are outlined in Table 3-2.

TWL-06-0156-001-A
Released: July 8, 2009

Table 3-2 BG Type

Method	BG Type	Characteristics
	3D	Can display graphics generated by the 3D Graphics Engine.
Character Method	Text	This is a character method BG. It is the only method that can handle character data defined with 16 colors. It can reduce consumption of VRAM, but cannot perform affine conversions.
Character Method	Affine	This is a character method BG that can perform affine conversions. It cannot perform character unit processing (HV Flips).
Character Method / Bitmap Method	Affine Extended BG	This method uses one of the following options at a time: ・A character BG that uses 256 colors × 16 palettes ・A 256-color bitmap BG ・A bitmap BG where colors can be directly specified
Bitmap Method	Large Screen 256-Color Bitmap	This is a large screen bitmap BG. It cannot be used together with other BGs, since it uses up the maximum amount of VRAM (512 kilobytes) for BGs on one layer. It can be used together with the 3D layer.

There are four layers each (BG0-BG3) for 2D Graphics Engine A and B. The BG Mode determines which BG type will be used for the layers.

There are different types of BG Mode for 2D Graphics Engine A and 2D Graphics Engine B. There are seven BG Modes to choose from for 2D Graphics Engine A, as listed in Table 3-3. There are six BG Modes to choose from for 2D Graphics Engine B, as listed in Table 3-4.

Table 3-3 BG Modes for 2D Graphics Engine A

BG Mode	BG0	BG1	BG2	BG3
0	Choose Text or 3D	Text	Text	Text
1			Text	Affine
2			Affine	Affine
3			Text	Affine Extended BG
4			Affine	Affine Extended BG
5			Affine Extended BG	Affine Extended BG
6	3D	×	Large Screen 256-Color Bitmap	×

Table 3-4 BG Modes for 2D Graphics Engine B

BG Mode	BG0	BG1	BG2	BG3
0	Text	Text	Text	Text
1			Text	Affine
2			Affine	Affine
3			Text	Affine Extended BG
4			Affine	Affine Extended BG
5			Affine Extended BG	Affine Extended BG

Note: Neither 3D nor Large-Screen 256-Color Bitmap cannot be selected as a BG type for 2D Graphics Engine B.

The settings and features for each BG layer differ depending on BG type. See Table 3-5 for details.

Table 3-5 BG Features

Method		Character BG					Bitmap BG
BG Type / Features & Specifications	3D	Text	Affine	Affine Extended BG: 265 Colors × 16 Palettes	Affine Extended BG: 256 Colors	Affine Extended BG: Direct Color	Large Screen 256-Color Bitmap
Screen Size	256×192	256×256 512×256 256×512 512×512	128×128 256×256 512×512 1024×1024	128×128 256×256 512×512 1024×1024	128×128 256×256 512×256 256×512	128×128 256×256 512×256 256×512	512×1024 1024×512
Number of Characters	×	1024	256	1024	×	×	×
Number of Colors / Palettes	262144	16／16 256／1 256／16	256／1	256／16	256／1	32768	256／1
Affine Conversion	×	×	○	○	○	○	○
Out-of-Area Processing	×	×	○	○	○	○	○
HV Flip	×	○	×	○	×	×	×
H-Scroll	○	○	○	○	○	○	○
V-Scroll	×	○	○	○	○	○	○
Mosaic	×	○	○	○	○	○	○
Priority	○	○	○	○	○	○	○

3.3.1 Screen Size

This is the screen size (in pixels) of the BG.

3.3.2 Number of Characters

This is the number of characters that can be defined within character data when the basic character is an 8×8 pixel.

3.3.3 Number of Colors / Palettes

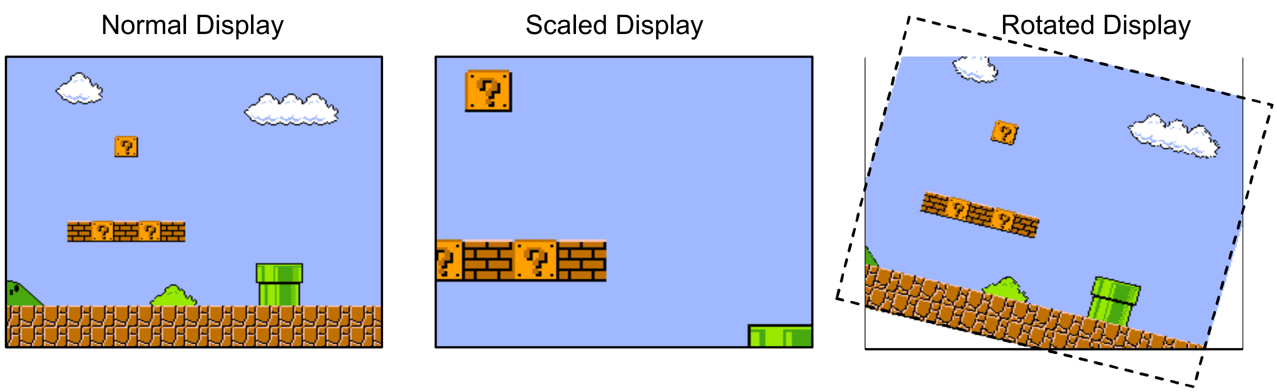
This is the number of colors and color palettes that can be selected. The BG types that do not use color palettes are indicated by the number of colors they display (in blue lettering in Table 3-5).

For details about the color palette, see section 3.4 Color Palette.

3.3.4 Affine Conversion

This feature deals with the rotated and scaled display of a BG screen. (See Figure 3-9.)

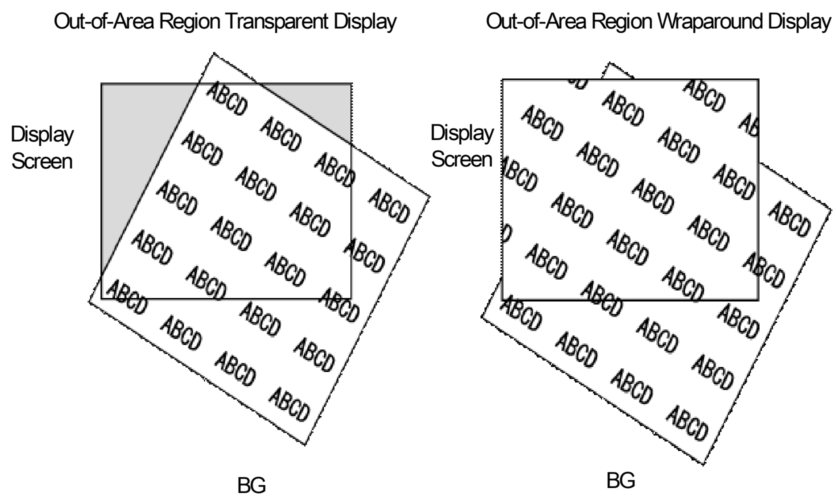
Figure 3-9 BG Affine Conversion



3.3.5 Out-of-Area Processing

If the displayable screen exceeds the BG screen due to affine conversion, the region that exceeds the BG screen can be made transparent or can be wrapped and displayed. (See Figure 3-10.)

Figure 3-10 BG Out-of-Area Processing



3.3.6 HV Flip

This feature allows individual BG character units to be displayed in reverse (either horizontally and/or vertically)

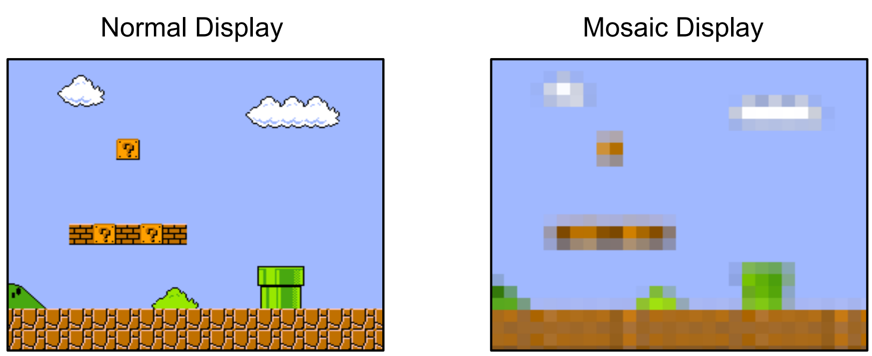
3.3.7 H-Scroll, V-Scroll

This feature is for horizontal and vertical scrolling of the BG.

3.3.8 Mosaic

This feature is for the display of mosaics. The number of pixels in the mosaic can be specified. (See Figure 3-11.)

Figure 3-11 BG Mosaic

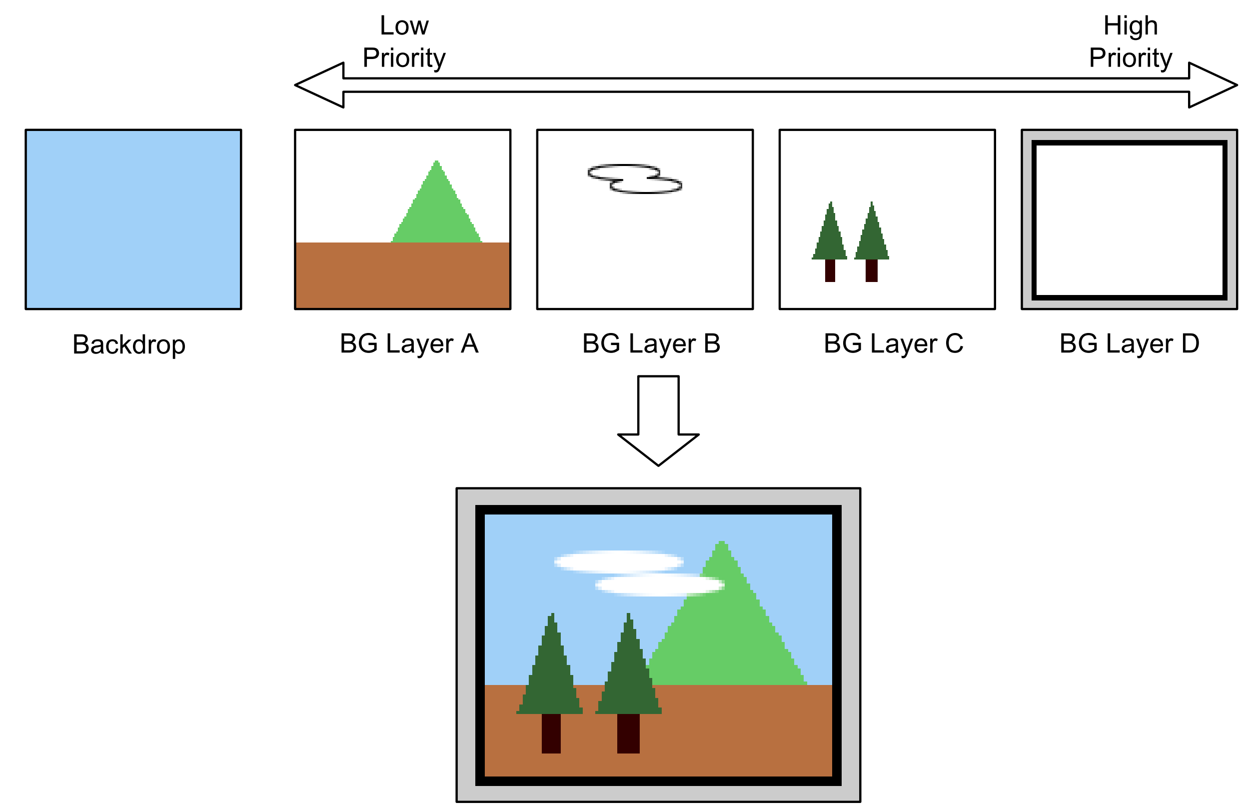


TWL-06-0156-001-A
Released: July 8, 2009

3.3.9 Priority

There are four levels of display priority. The BG layer with the highest priority is displayed in front. (See Figure 3-12.)

Figure 3-12 BG Priority



3.4 Color Palette

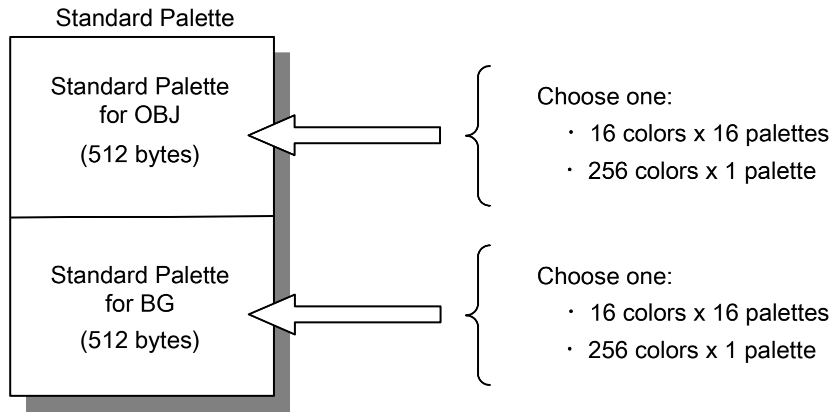
There are two categories of color palettes.

3.4.1 Standard Palette

The standard palette is stored in the palette-exclusive RAM in the 2D Graphics Engine.

There is 1 kilobyte of RAM for 2D Graphics Engine A and 1 kilobyte of RAM for 2D Graphics Engine B. The standard palette is divided into a part for OBJ (512 bytes) and a part for BG (512 bytes). There are two palette types that can be selected for each part. (See Figure 3-13.)

Figure 3-13 Standard Palette



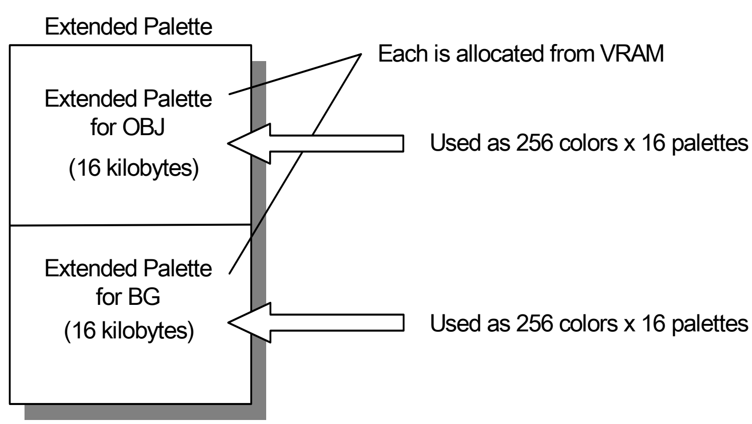
3.4.2 Extended Palette

To use the extended palette, VRAM must be allocated for extended palette use.

The allocated extended palette is used as 256 colors × 16 palettes. (See Figure 3-14.)

The extended palette can only be applied to OBJs and BGs in 256 Color × 16 Palette Mode.

Figure 3-14 Extended Palette



Keep in mind that the color with index number 0 in each palette is always used as a transparent color, in all color palettes.

3.5 Windows

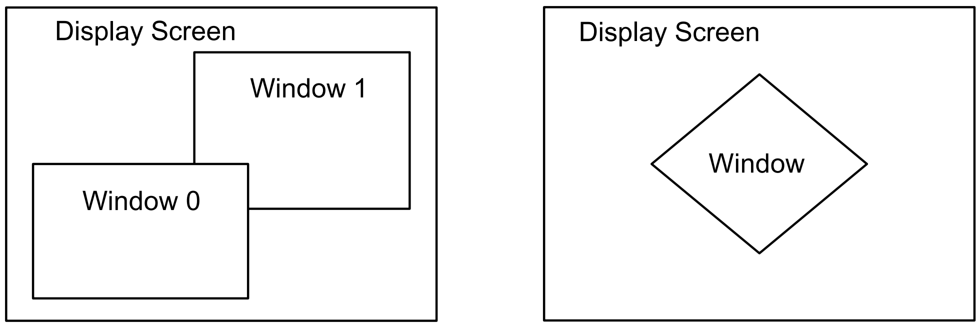
The window feature can control three kinds of windows: Window 0, Window 1, and the OBJ Window.

3.5.1 Window 0, Window 1 (See Figure 3-15.)

These are rectangular windows.

During the H-Blank interval, the window region can be reshaped by changing its coordinates.

Figure 3-15 Window Examples

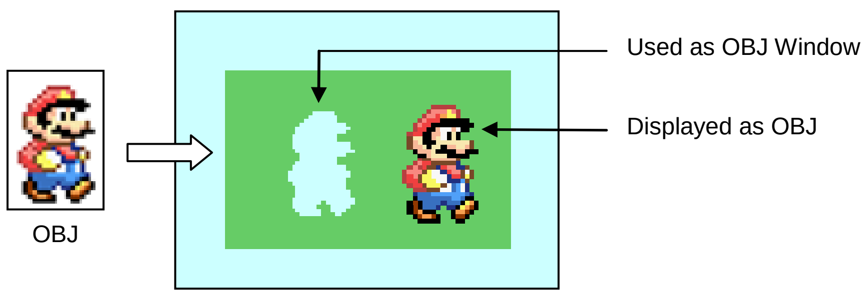


3.5.2 OBJ Window

An OBJ specified with the OBJ window flag can be used as a window.

Unlike windows 0 and 1, this window can take on any shape. (See Figure 3-16.)

Figure 3-16 OBJ Window



The inside and outside of each window can be set respectively in the following manner:

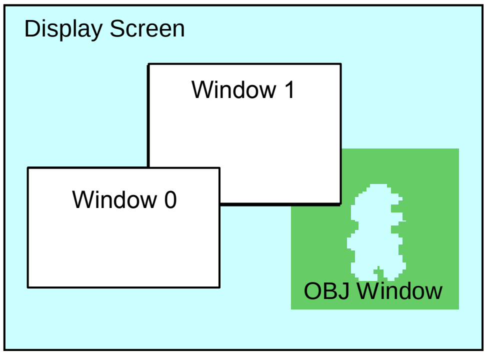
- Whether or not each BG layer or OBJ is displayed
  The area where the BG and OBJ display can be limited.
- Whether or not color special effects can be applied
  The area where color special effects are applied can be limited.

3.5.3 Window Priority

Window 0 always has priority over Window 1. The OBJ Window has the lowest priority.

Window priority cannot be changed. (See Figure 3-17.)

Figure 3-17 Window Priority



3.6 Color Special Effects

The special effects below can be applied to BGs, OBJs, and backdrop layers.

- α-Blending

Conducts color blending by using blending coefficients on two specified layers.

- Increase Brightness

Increases the brightness on a specified layer.

- Decrease Brightness

Decreases the brightness on a specified layer.

3.7 Differences from GBA

Following are the differences compared with GBA and the improvements to the 2D graphics features:

3.7.1 OBJ

- A bitmap OBJ has been added.
- An extended palette (256 colors × 16 palettes) has been added.
- The number of characters that can be drawn simultaneously has increased. (However, the means of data positioning is limited.)
- More OBJs can be displayed on one line and OBJs do not readily disappear.

3.7.2 BG

- The number of BG types has increased. Specifically, the Affine Conversion Extended BG and the Large-Screen 256-Color Bitmap BG have been added.
- The number of BG Modes has increased.
    - Regardless of BG type, 4 BG layers can now always be displayed.
    - The character method BG and bitmap method BG can now be blended.
- An extended palette (256 colors × 16 palettes) has been added.

4 3D Graphics

4.1 The Rendering Process

3D graphics functionality is divided into two stages: the GE (Geometry Engine) stage and the RE (Rendering Engine) stage.

A rough outline of the rendering process using GE and RE is shown in Figure 4-1.

Figure 4-1 Graphics Rendering Process

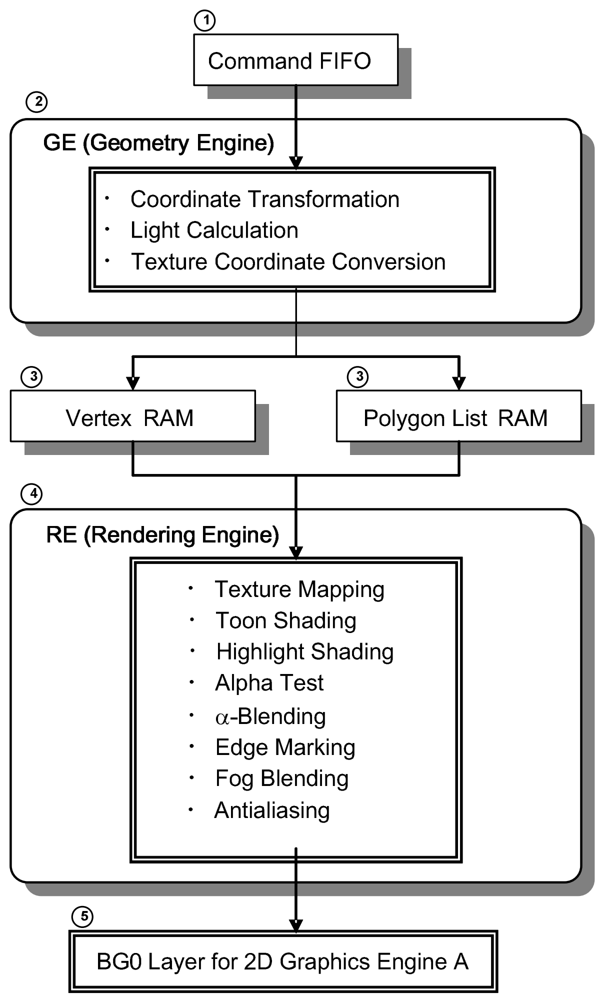
① The rendering command written to command FIFO by the CPU is sent to the GE.

② The GE conducts vertex calculation based on the rendering command it was sent.

③ The vertex information created by the GE is stored in exclusive RAM and sent to the RE.

④ The RE conducts pixel rendering based on the vertex information received from the GE.

⑤ The image rendered by the RE is displayed on the BG0 layer.



4.2 GE (Geometry Engine)

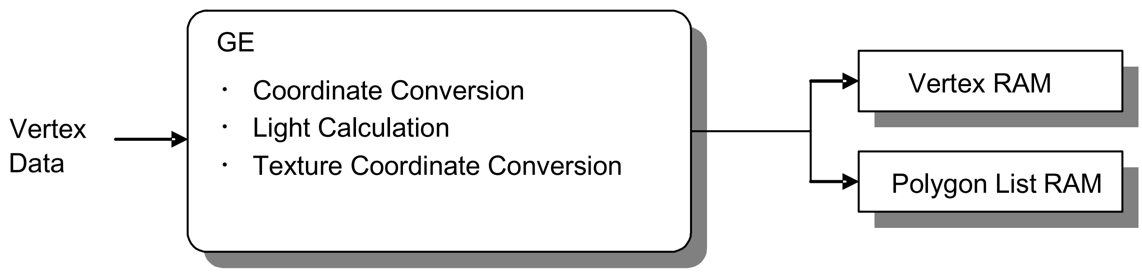
4.2.1 GE Functionality

There are three tasks performed by the GE.

- Coordinate Conversion
- Light Calculation
- Texture Coordinate Conversion

The GE carries out these processes on the vertex data it is sent. The completed vertex information is stored in dedicated RAM referred to as Vertex RAM and Polygon List RAM. (See Figure 4-2.)

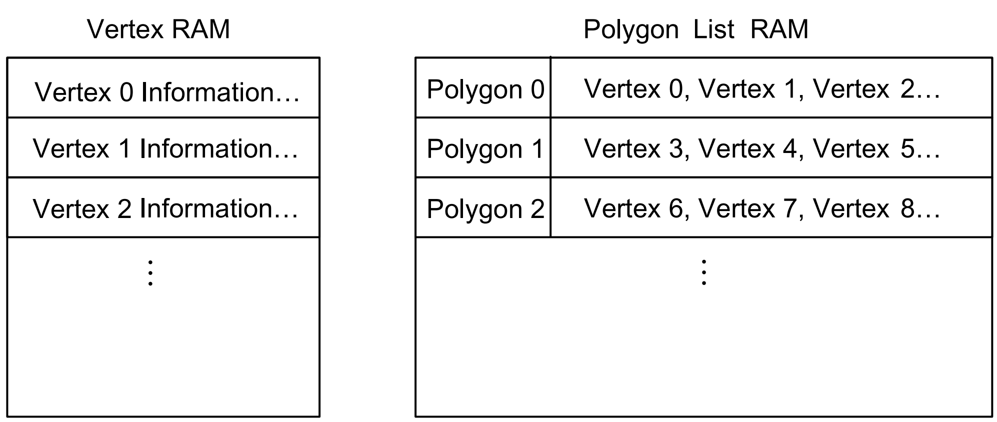
Figure 4-2 Processes Carried Out by the GE



Vertex information is stored in Vertex RAM after processing.

The information for how each vertex connects to form a polygon is stored in Polygon List RAM. (See Figure 4-3.)

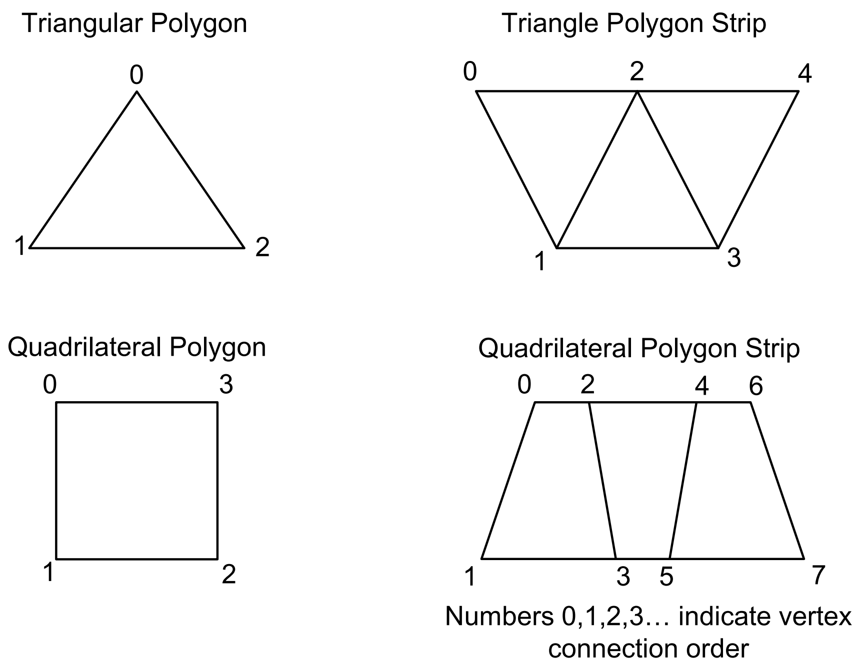
Figure 4-3 Overview of Vertex RAM and Polygon List RAM



There are four types of polygons that can be rendered. (See Figure 4-4.)

- Triangular Polygons
- Triangle Polygon Strips
- Quadrilateral Polygons
- Quadrilateral Polygon Strips

Figure 4-4 Polygon Shapes



Since there is a limit to the number of vertices (6144) that can be stored in Vertex RAM, it is necessary to plan for efficiency using these shapes so that most polygons will be rendered with as few vertices at possible.

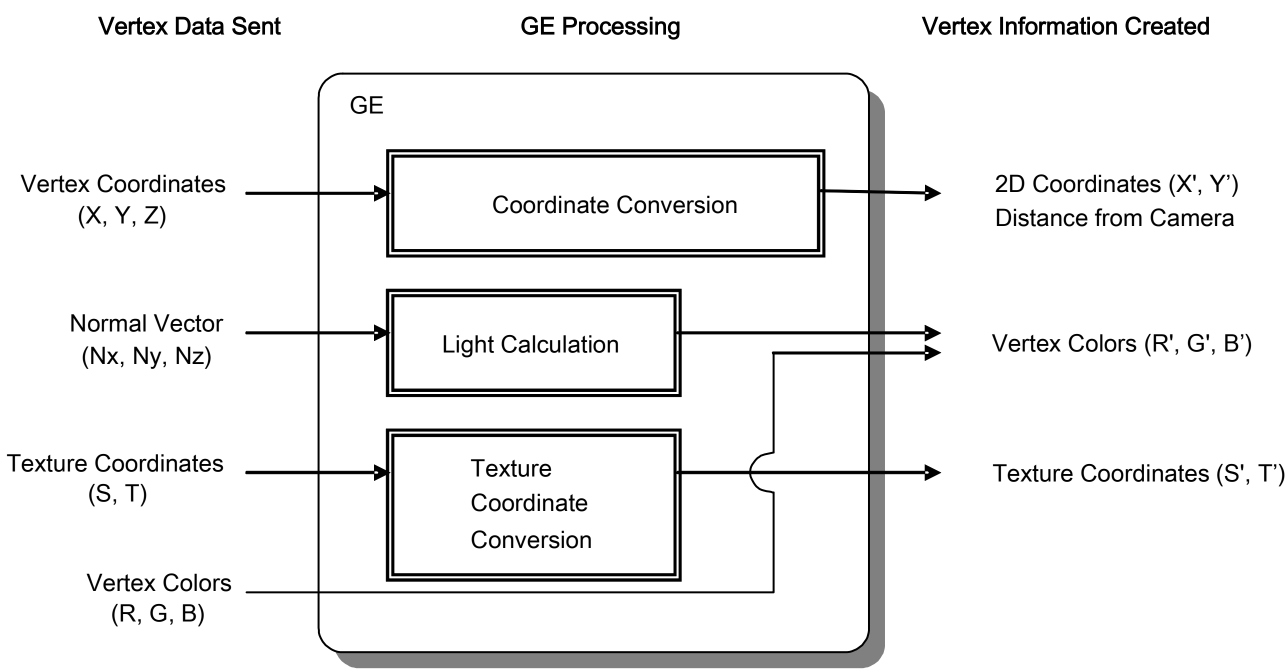
4.2.2 Vertex Data Processed by the GE

There are four kinds of vertex data that can be sent to the GE.

- Vertex Coordinates (X, Y, Z)
- Normal Vectors (Nx, Ny, Nz)
- Texture Coordinates (S, T)
- Vertex Colors (R, G, B)

The GE processing and data flow for each kind of vertex data sent is as shown in Figure 4-5.

Figure 4-5 Vertex Data Sent to GE and Their Process Flows



4.2.3 Coordinate Conversion

There are two kinds of coordinate conversions carried out by the GE. (See Figure 4-6.)

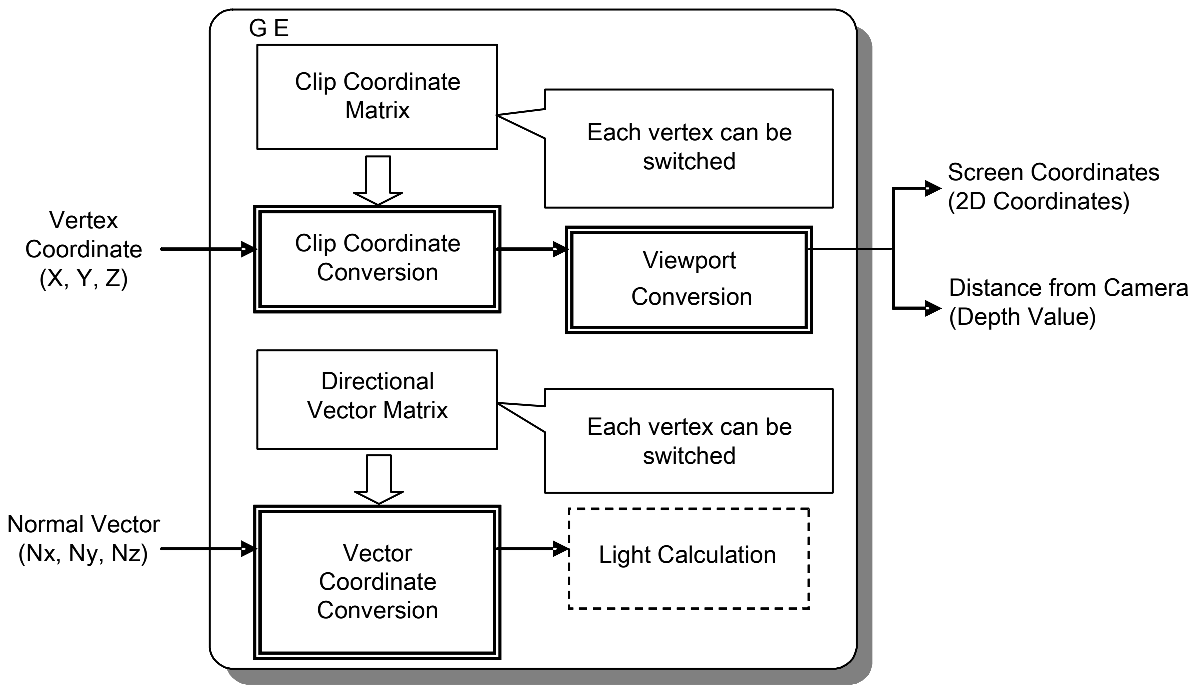
- Vertex Coordinate Clip Coordinate Conversion

  Vertex Coordinates are converted to onscreen 2D coordinates and the depth value expressing the camera distance. Coordinate transformation and clipping is conducted by the clip coordinate matrix. (Details about the clip coordinate matrix have been omitted from this manual.)

- Normal Vector Coordinate Conversion

  The normal vector is converted into a vector with camera coordinates by the directional vector matrix, and then used in light calculation. (Details about the directional vector matrix have been omitted from this manual.)

Figure 4-6 Coordinate Conversion



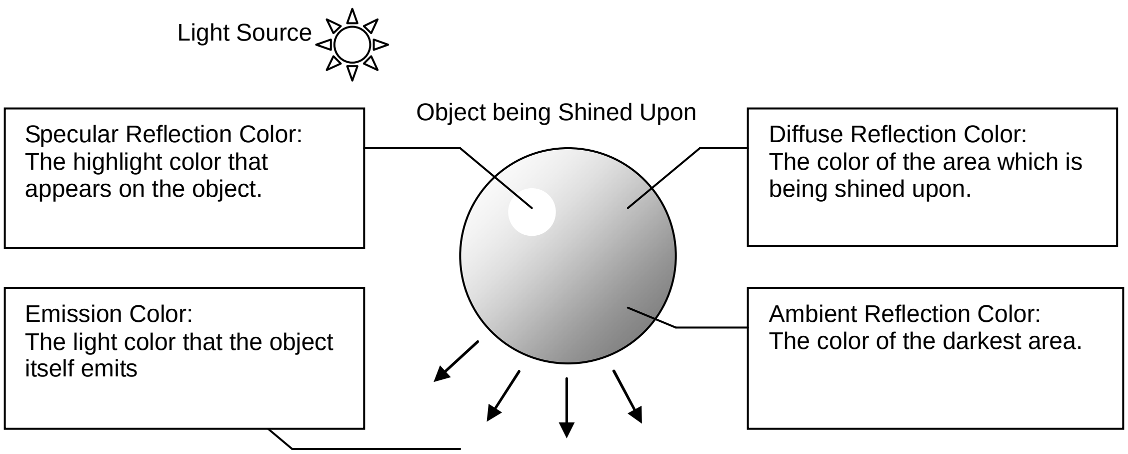
The settings for the clip coordinate matrix and the directional vector matrix can be changed for each vertex.

By switching these conversion matrices at each vertex, different polygon shapes can be expressed (with envelope, skinning, etc.).

4.2.4 Light Calculation

In general terms, light calculation is functionality that calculates the influence of a light source on an object's (polygon's) material, and expresses the color and shading of that object. The special qualities of object materials are shown below. (See Figure 4-7.)

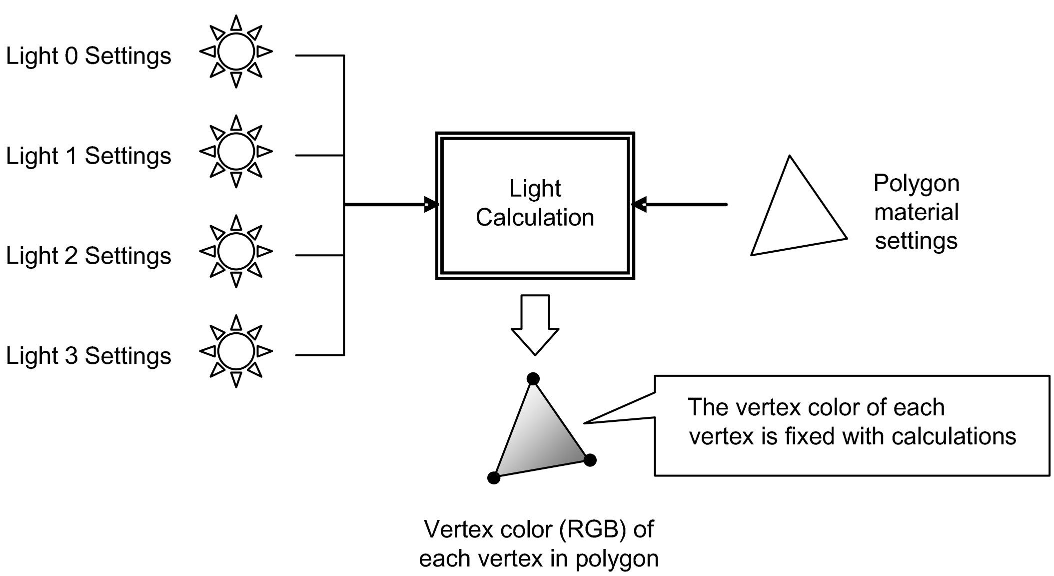
Figure 4-7 Light Source and Object Material



DS supports four parallel light sources (Lights 0–3).

Light calculations are executed based on these four light's settings. The vertex color of each of the polygon's vertices is set as the calculation result. (See Figure 4-8.)

Figure 4-8 Light Calculation



4.2.4.1 Light Settings

Each light can be set according to the following parameters.

- A vector that expresses light direction (Lx, Ly, Lz)
- The light color (RGB)

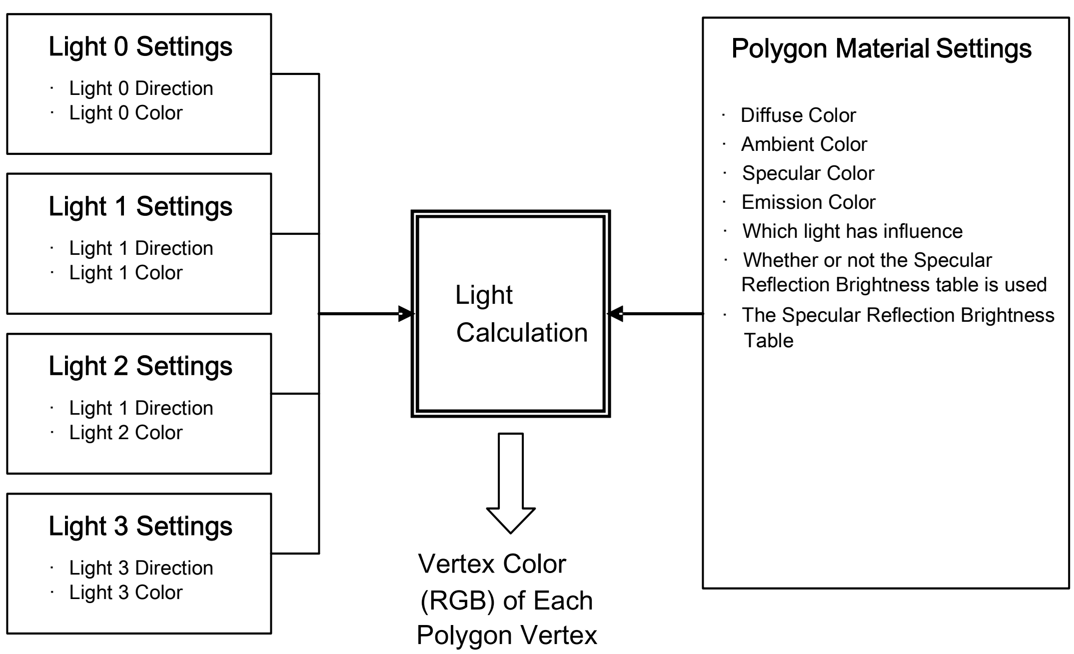
4.2.4.2 Polygon Material Settings

Polygon materials can be set according to the following parameters.

- Diffuse color (RGB)
- Ambient color (RGB)
- Specular color (RGB)
- Emission color (RGB)
- Which lights influence the object (any combination of Lights 0–3)
- Whether the Specular Reflection Brightness table is used when the specular component is calculated.
- The Specular Reflection Brightness Table

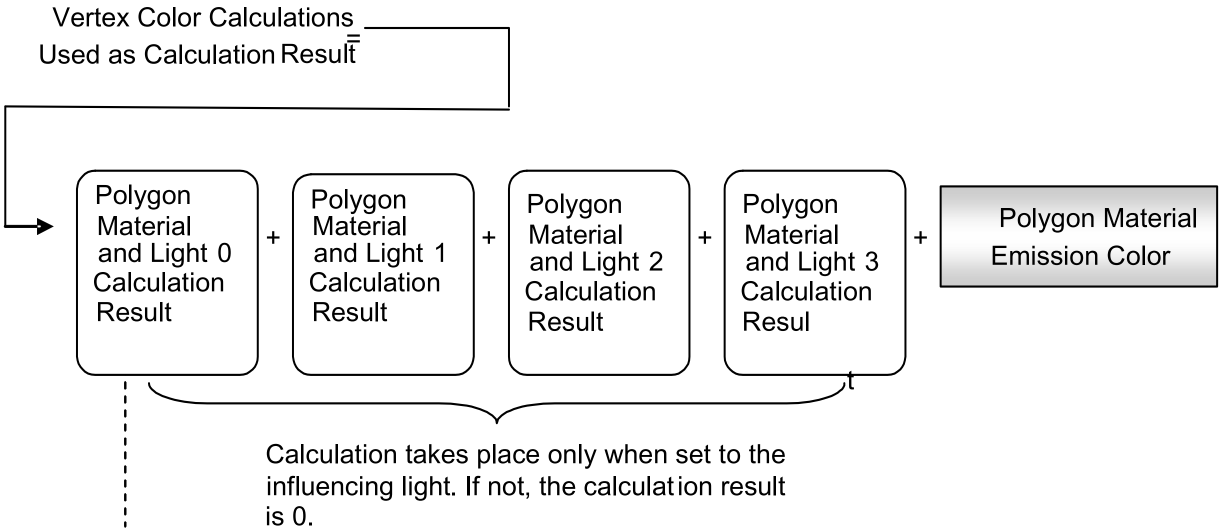
Taken as a whole, the paramenters for setting light and polygon materials are used to perform light calculations as shown below. (See Figure 4-9.)

Figure 4-9 Parameters Used in Light Calculation



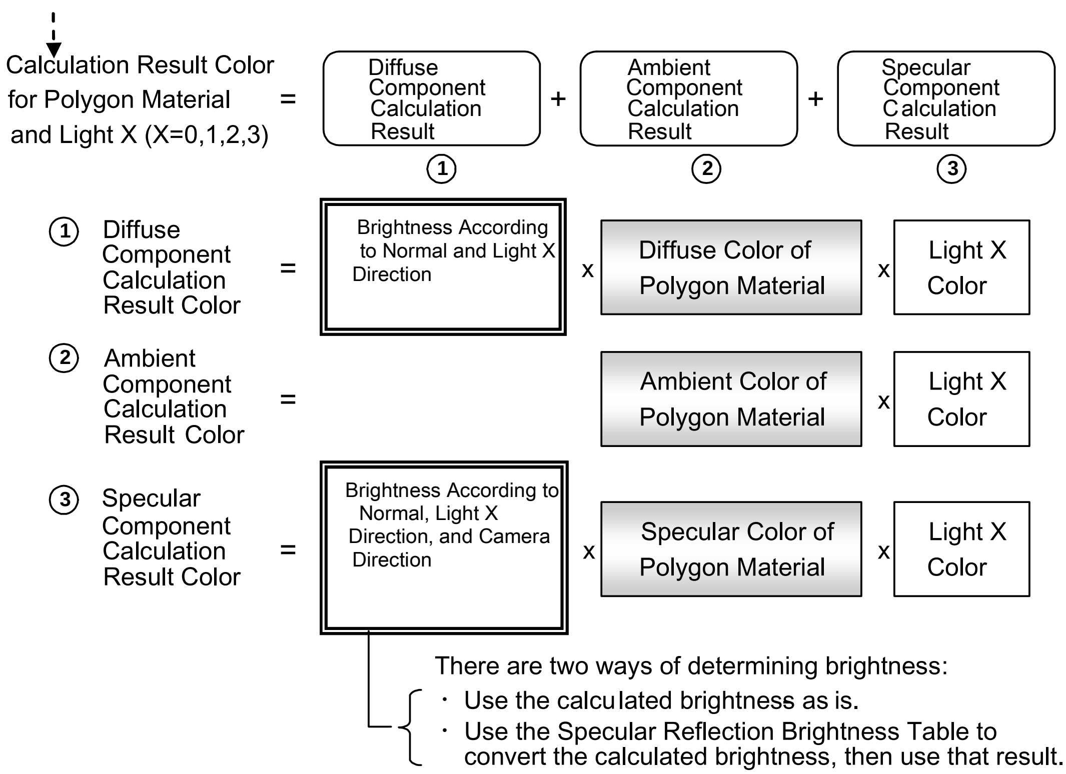
4.2.4.3 Light Calculation Method

Light calculation is conducted with the following equation for each RGB component:



Calculations for the polygon material and each light is only conducted when it is configured to be influenced by that light. When there is no received light influence, the emission color alone is used as the vertex color result.

Following are the details of polygon material calculation and calculation for each light:



Each of the component calculations (1, 2, and 3) must be carried out.

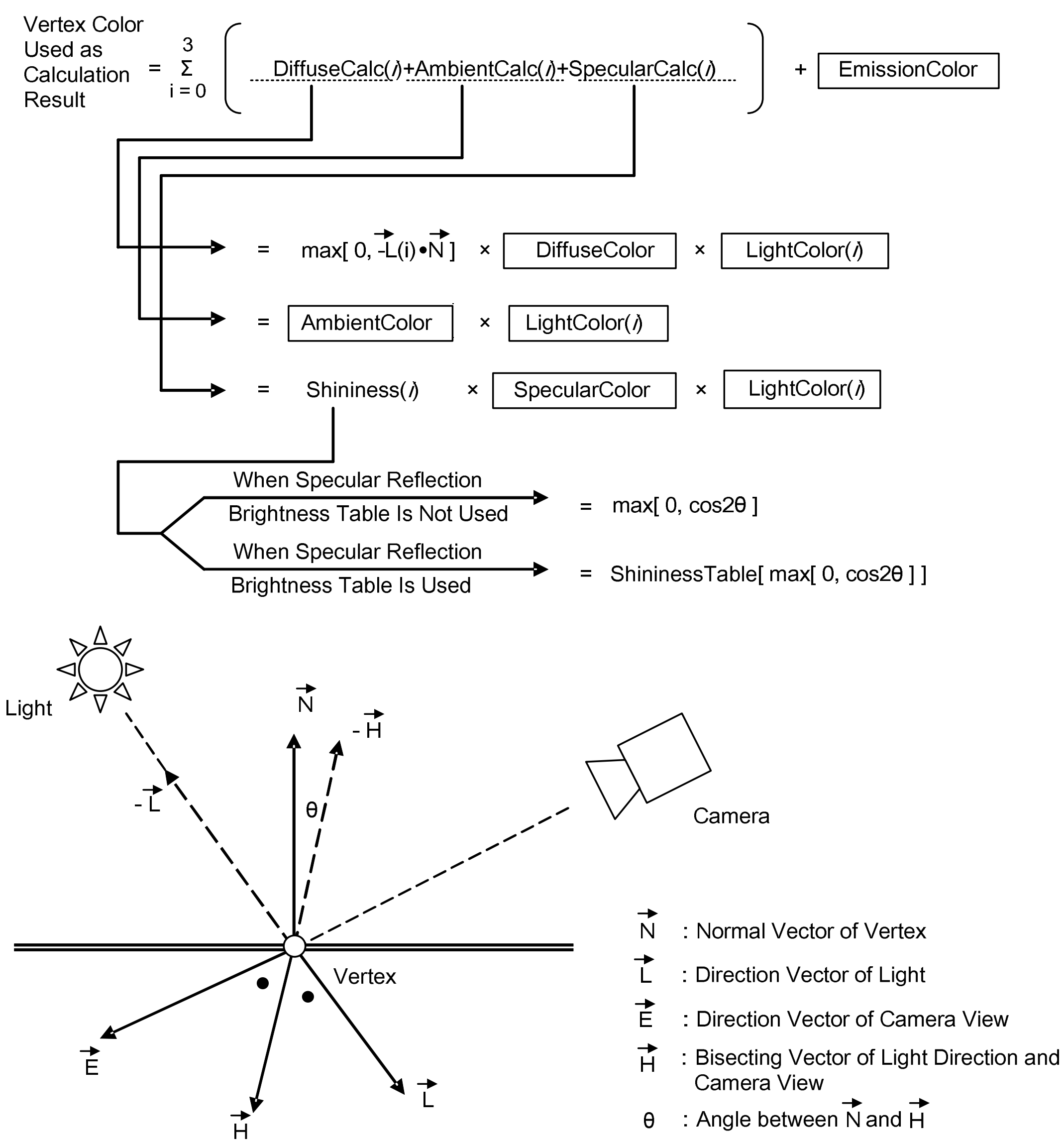
If a component is not needed, the color applied to the polygon material (diffuse color, ambient color, specular color) must be set to black (R = G = B = 0).

4.2.4.4 Equation Details (for Programmers)

The mathematical details for the equations used in light calculation are shown in Figure 4-10.

i represents the light number, and DiffuseCalc(*i*), AmbientCalc(*i*), and SpecularCalc(*i*) are calculated only when the influence of light *i* is received. Also, all vectors used in the calculation method are unit vectors.

Figure 4-10 Mathematical Details of Light Calculation Method



4.2.4.5 Specular Reflection Brightness Table

The Specular Reflection Brightness Table is a table for converting the specular reflection brightness used by the specular component calculation.

The brightness of specular highlights can be controlled using the Specular Reflection Brightness Table.

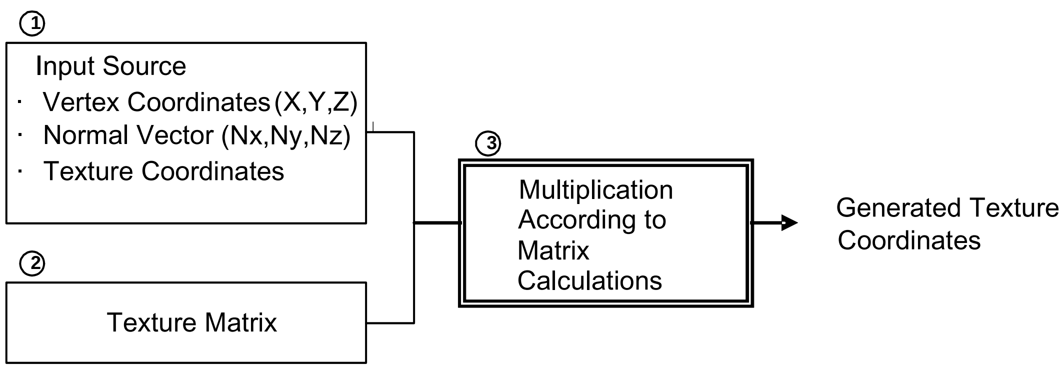
4.2.5 Texture Coordinate Conversion

Texture coordinate conversion generates the texture coordinates for textures applied to polygons.

The texture coordinate conversion flow is provided below. (See Figure 4-11.)

1. The input source is designated from among the following three types of vertex data.
    - Vertex Coordinates (X, Y, Z)
    - Normal Vectors (Nx, Ny, Nz)
    - Texture Coordinates (S, T)
2. The matrix for texture coordinate conversion (the texture matrix) is configured.
3. The input sources designated in step 1 and the texture matrix designated in step 2 are multiplied according to matrix calculations, and the texture coordinate is generated.

Figure 4-11 Texture Coordinate Conversion



By putting together the input source and the texture matrix, a variety of texture placement methods can be expressed.

The following is an example of such an expression.

- Texture Affine Conversion (See Figure 4-12 and Figure 4-13)

Figure 4-12 Texture Rotation

Input Source : Texture Coordinates(S,T)

Texture Matrix : Rotation Matrix

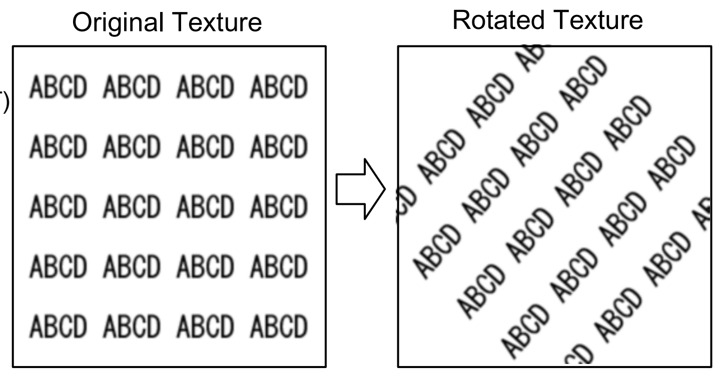
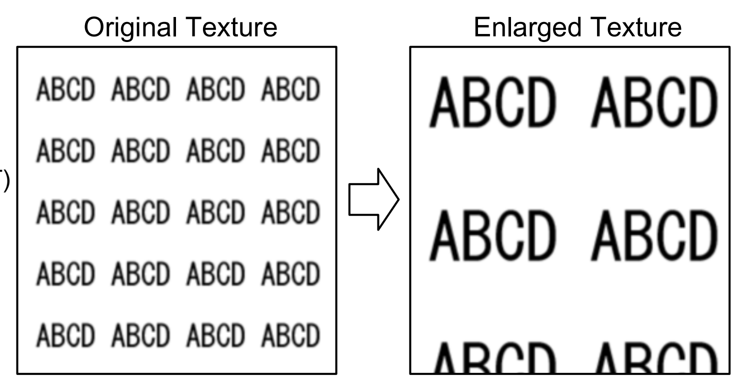


Figure 4-13 Texture Enlargement

Input Source: Texture Coordinates(S,T)

Texture Matrix: Enlarge Matrix



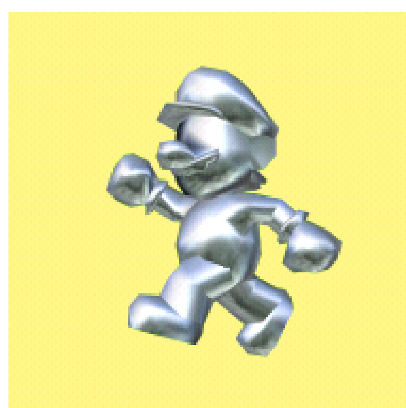
- Texture Environment Mapping (See Figure 4-14)

Figure 4-14 Texture Environment Mapping

Input Source: Normal Vector (Nx,Ny,Nz)

Texture Matrix: Directional Vector Matrix

Allows for the reflection of the surrounding environment as a texture map.



This figure represents an environment-mapped image. Actual display may differ.

4.2.6 Polygon Attributes

Though not directly handled during GE calculation, each polygon can be set with the following attribute values during the RE rendering process.

- Polygon ID

  The polygon ID can be set to any value from 0 to 63 (these are 6-bit values).

- Polygon α Value

  The polygon α value represents its degree of translucency, and can be set to any value from 0 to 31 (these are 5-bit values).

  The following handling differences depend on the set values.

  Polygon α Value
  - 0 ・・・ Rendered as a wireframe polygon
  - 1 – 30 ・・・ Handled as a translucent polygon
  - 31 ・・・ Handled as an opaque polygon

- Polygon Modes

  Polygon Modes are used when mixing polygon colors and texture colors in the RE.

  The four mode types below can be selected.

  For more information on each mode, see section 4.3.4 Blending Polygon Color and Texture Color.

  Polygon Modes
  - Decal Mode
  - Modulation Mode
  - Toon Shading
  - Highlight Shading

- Use / Do not use fog

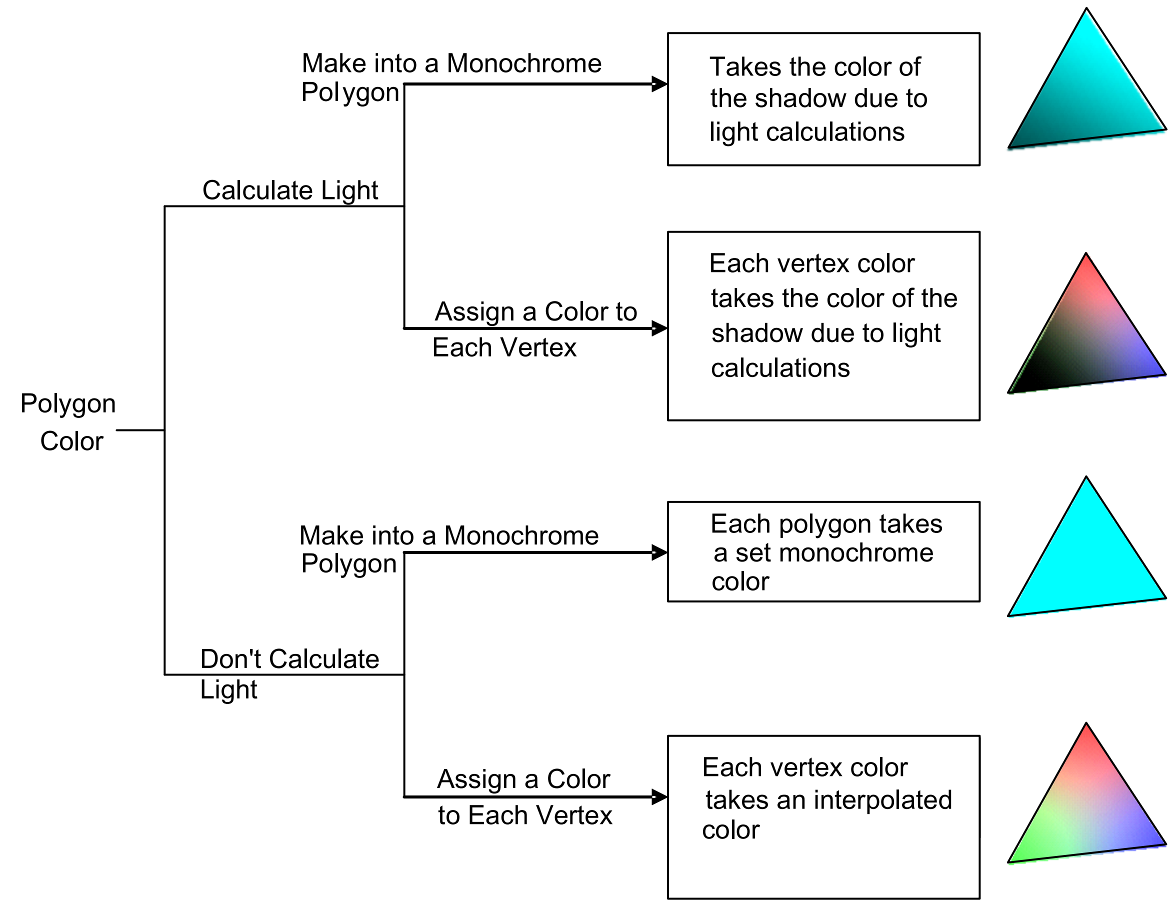
  Allows for the selection of whether fog is used.

  For more information on fog, see section 4.3.8 Fog Blending.

4.2.7 Supplemental 1: Deciding a Polygon's Color

Broadly speaking, there are four methods for deciding a polygon's color. These methods are indicated below. (See Figure 4-15.)

Figure 4-15 How to Decide a Polygon's Color



The above information is based on actual test results using the development hardware.

Note: For the second of the four methods above (*Calculate Light* > *Assign a Color to Each Vertex*), if the polygon vertex normals are the same, then the same command (Normal command) must be sent multiple times to the GE (which increases processing time.) Care is therefore needed when implementing this method.

4.3 RE (Rendering Engine)

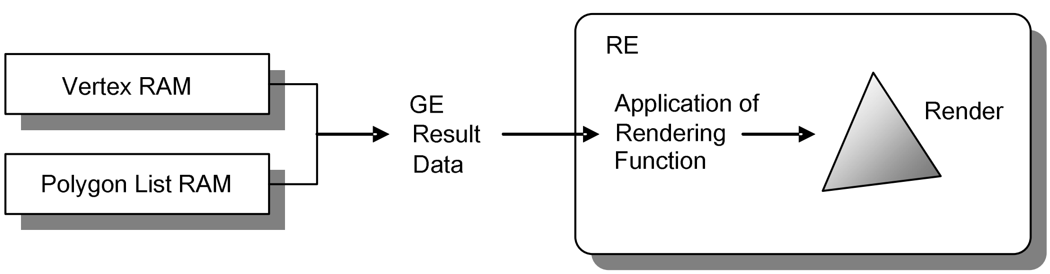
4.3.1 RE Functionality

The RE renders polygons based on the results of what was created by the GE.

When rendering polygons, the following variety of graphics functions can be applied. (See Figure 4-16.)

- Texture Mapping
- Combining Polygon and Texture RGBAs
  - Decal Mode
  - Modulation Mode
  - Toon Shading
  - Highlight Shading
- Alpha Test
- α-Blending
- Edge Marking
- Fog Blending
- Antialiasing

Figure 4-16 Processes performed by the RE



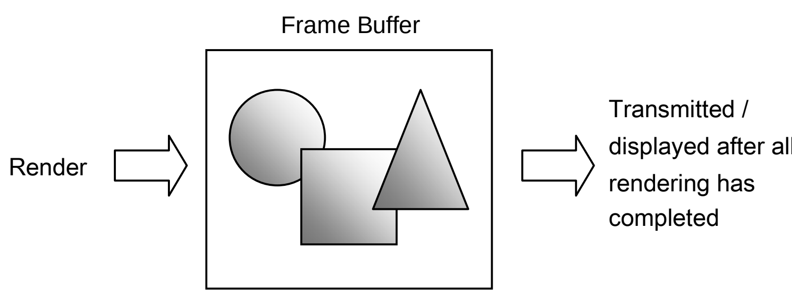
4.3.2 Line Buffer Method and Frame Buffer Method

The rendering method used by the RE differs from the more common method of rendering to a frame buffer. The RE uses the line buffer method, which draws one line at a time from the top of the screen down.

4.3.2.1 General Frame Buffer Methods

Image data is transmitted for display after all rendering to the frame buffer has completed. (See Figure 4-17.) If rendering does not finish within one frame, a processing error (where the same image is repeatedly displayed) will occur.

Figure 4-17 General Frame Buffer Methods

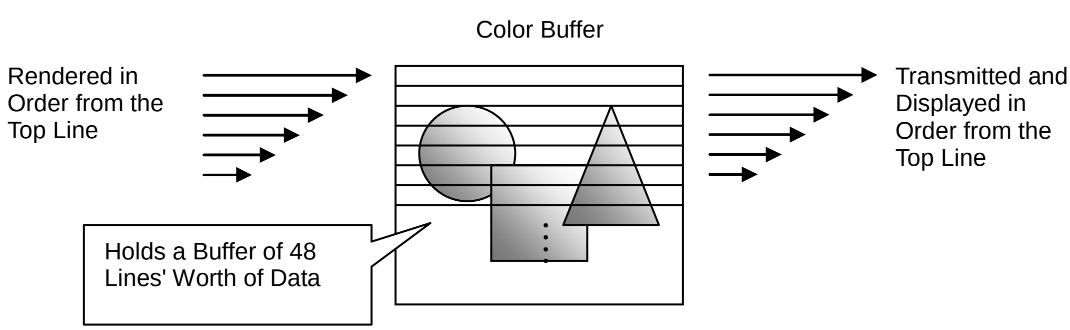


4.3.2.2 Line Buffer Method

Rendering is carried out in order, beginning with the top line in the color buffer and processing image data one line at a time. (See Figure 4-18.)

The RE has a 48-line color buffer. Line data is written to and stored in the color buffer before it is displayed.

Figure 4-18 Line Buffer Method



If the number of rendered polygons and pixels in one line exceeds the RE's processing capability, the rendering will not be displayed in time, and the displayed image will be corrupted.

4.3.3 Texture Mapping

The following settings can be applied to each texture.

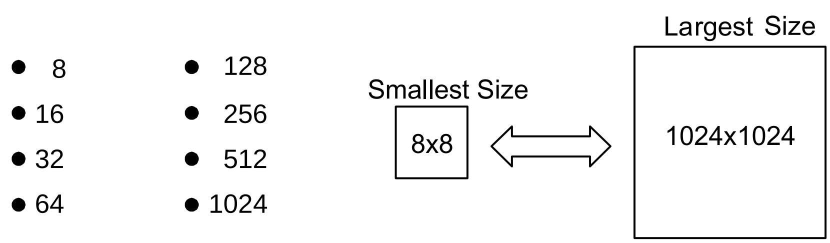
- Size (Width × Height)
- Format and Palette Color 0 Handling Method
- Repeat Method

4.3.3.1 Size (Width × Height)

Texture width and height can be selected from the eight sizes shown in Figure 4-19.

Width and height do not have to be equal; rectangular sizes are also possible.

Figure 4-19 Texture Size



4.3.3.2 Format and Palette Color 0 Handling Method

There are seven different types of texture format:

- 4-Color Palette Texture
- 16-Color Palette Texture
- 256-Color Palette Texture
- 4 × 4 Texel Compressed Texture
- A5I3 Translucent Texture
- A3I5 Translucent Texture
- Direct Color Texture

With the exception of the direct color texture, all formats are color index methods that use the color palette. The format of each entry in the color palette is RGB 555. There is no α component.

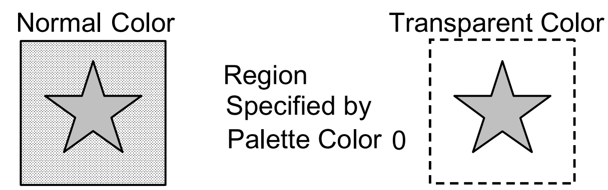
4-Color, 16-Color, and 256-Color Palette Textures

These are each color index formats that allow the selection of 4, 16, or 256 colors.

For these three formats, there are two ways that palette color 0 (the index 0 color on the color palette) can be handled. (See Figure 4-20.)

Figure 4-20 Palette Color 0 Handling Method

- Palette Color 0 Handled as a Normal Color
- Palette Color 0 Handled as a Transparent Color (no color)



4 × 4 Texel Compressed Texture

This is a color index format where data is compressed into 4 × 4 texels.

The details of this compression algorithm have been omitted from this manual. (See the *TWL Programming Manual* for more information).

A5I3 Translucent Texture

This is a format where the color is selected from an 8-color (3-bit) palette, and the α is directly specified using a 5-bit α value without using the palette. The 5-bit α value allows 32 levels of translucency to be expressed.

A3I5 Translucent Texture

This is a format where the color is selected from a 32-color (5-bit) palette, and the α value is directly specified using a 3-bit α value without using the palette. The 3-bit α value allows 8 levels of translucency to be expressed.

Direct Color Texture

This is the only format that is not a color index method. Color is specified with RGBA 5551 (16-bit).

This texture has a 1-bit α value, so it can display with transparency

A comparison of each format is indicated below. (See Table 4-1.)

Table 4-1 A Comparison of Texture Formats

Format	Number of Selectable Colors	Number of Bits per Texel	Can Display No Color	Can Display Translucency
4-Color Palette Texture	4	2	○*	×
16-Color Palette Texture	16	4	○*	×
256-Color Palette Texture	256	8	○*	×
4 × 4 Texel Compressed Texture	4 (For Each 4 × 4 Texel)	3	○	×

Format	Number of Selectable Colors	Number of Bits per Texel	Can Display No Color	Can Display Translucency
A5I3 Translucent Texture	8	8 (Color: 3-bit α: 5-bit)	○	○ (32 levels)
A3I5 Translucent Texture	32	8 (Color: 5-bit α: 3-bit)	○	○ (8 levels)
Direct Color Texture	32768	16 (RGBA5551)	○	×

* When Palette Color 0 is set as a translucent color

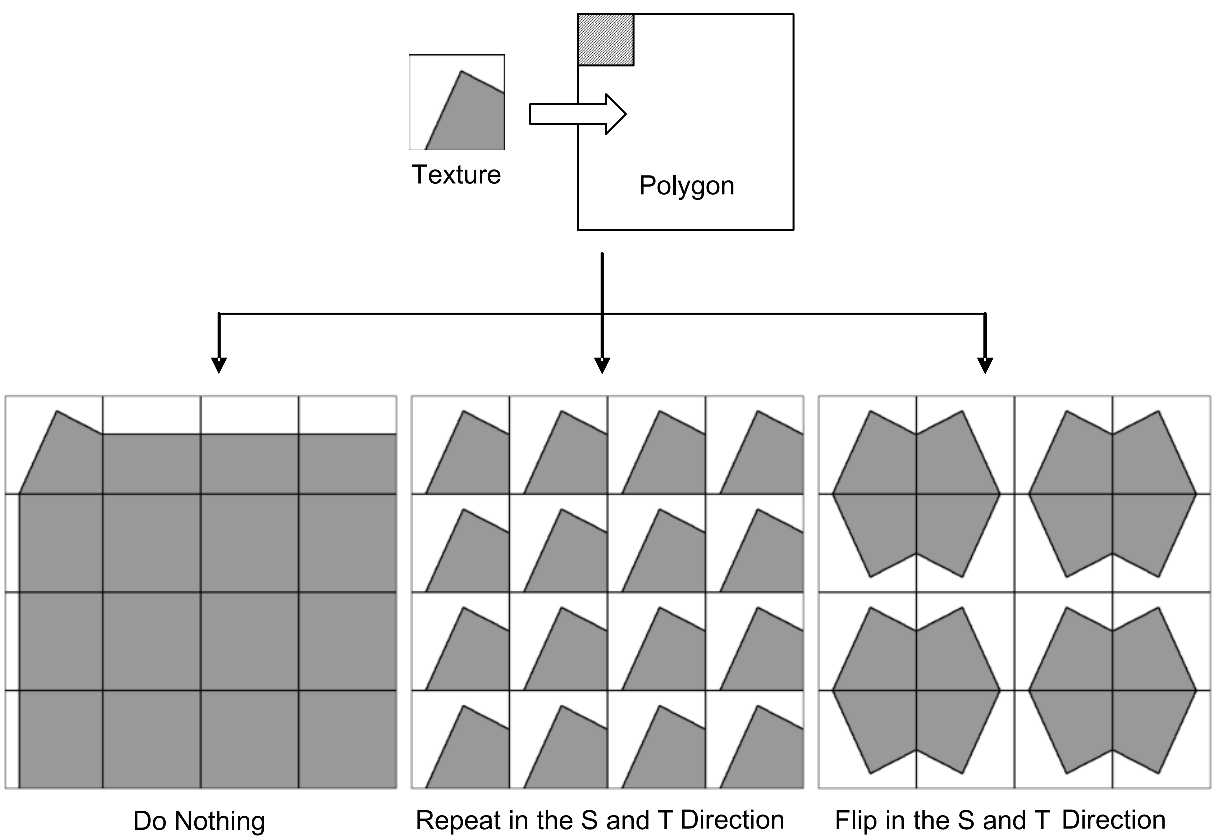
4.3.3.3 Flip and Repeat Methods

This specifies the draw method for regions where the texture coordinates exceed the texture size.

The following three kinds of selections can be made for the S and T directions. (See Figure 4-21.)

- Do Nothing — Texture edge color is stretched (also known as *clamp*ing).
- Repeat — Texture is repeated.
- Flip — Texture is reversed and repeated (known as a "mirror").

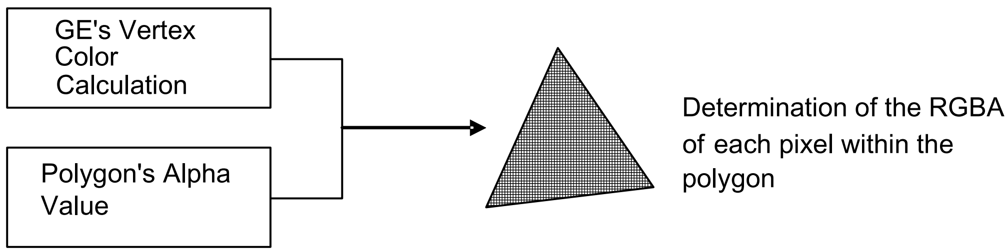
Figure 4-21 Texture Repeat Method



4.3.4 Blending Polygon Color and Texture Color

The RGBA values for each pixel within a polygon are calculated with the GE's vertex color calculation (RGB) and the polygon's α value. (See Figure 4-22.)

Figure 4-22 RGBA of Each Pixel Within a Polygon



The calculated RGBA values for each pixel are blended with the texture RGBA.

There are four ways of blending, depending on the polygon mode set in the polygon.

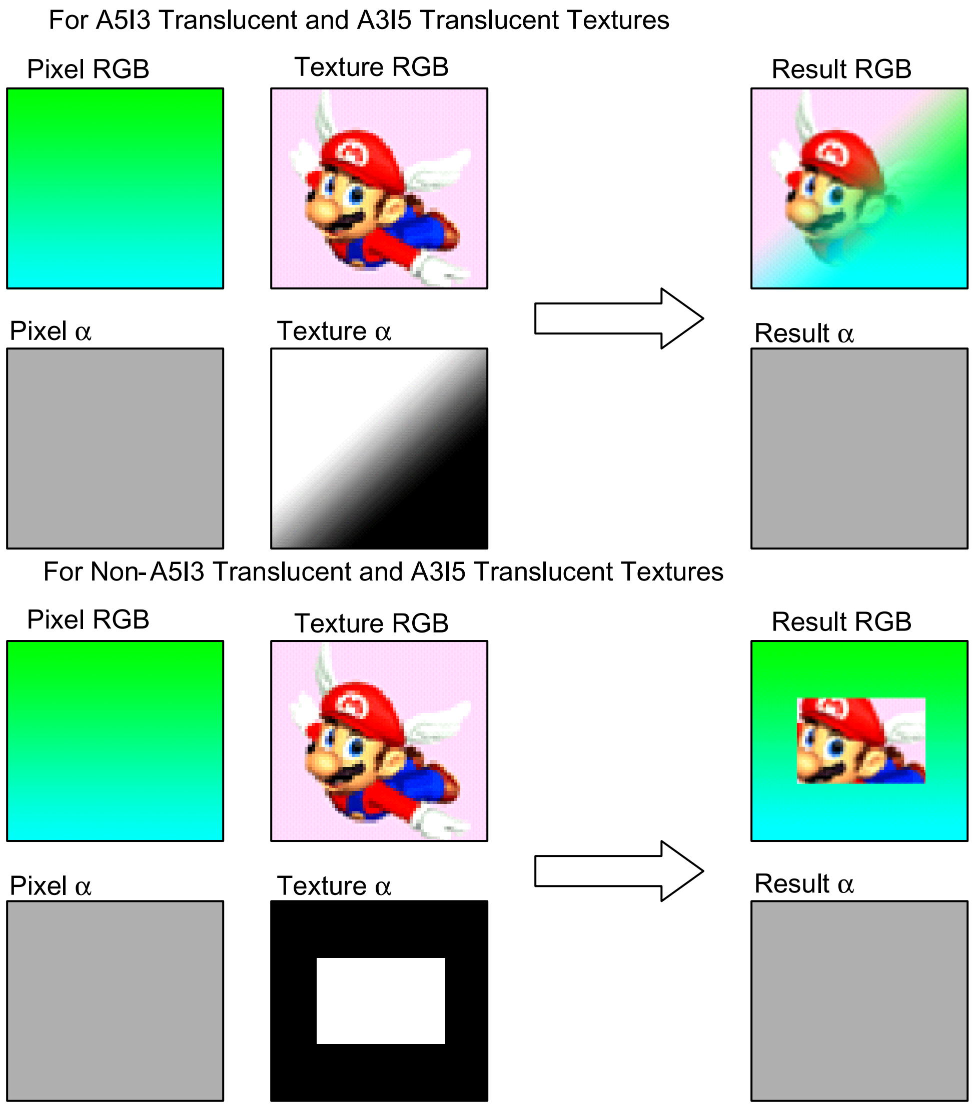
- Decal Mode
- Modulation Mode
- Toon Shading
- Highlight Shading

4.3.4.1 Decal Mode

Decal Mode makes the texture α value into a blend ratio, and blends each pixel's RGB and texture's RGB. For the α component, each pixel's α value is output as is. (See Figure 4-23.)

Figure 4-23 Decal Mode

RGB of Blend Result	=	Blended RGB of each pixel's RGB and texture's RGB using the texture α value
α Value of Blend Result	=	α value of each pixel



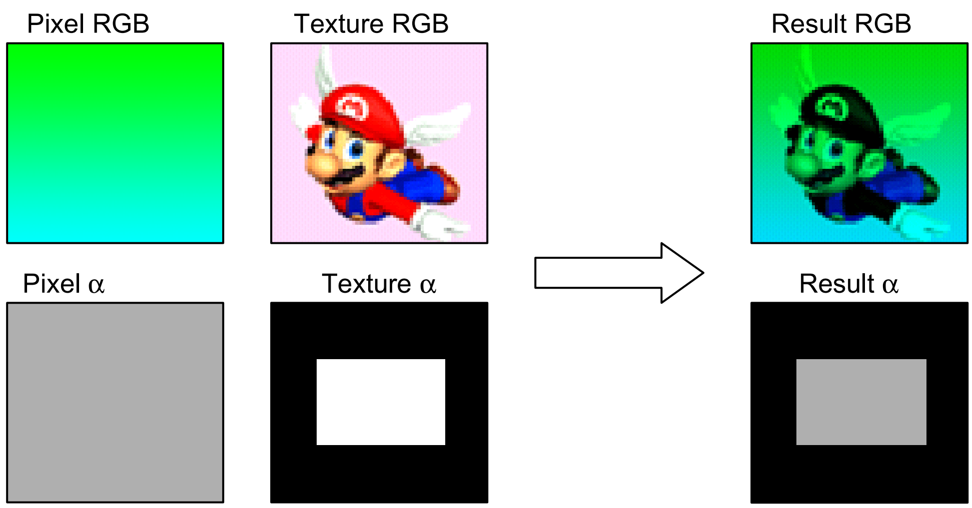
4.3.4.2 Modulation Mode

Modulation Mode multiplies the RGBA values of each pixel and texture, then blends them, as shown in Figure 4-24.

Figure 4-24 Modulation Mode

RGB of Blend Result = RGB of Each Pixel × Texture RGB

α Value of Blend Result = α Value of Each Pixel × Texture α Value



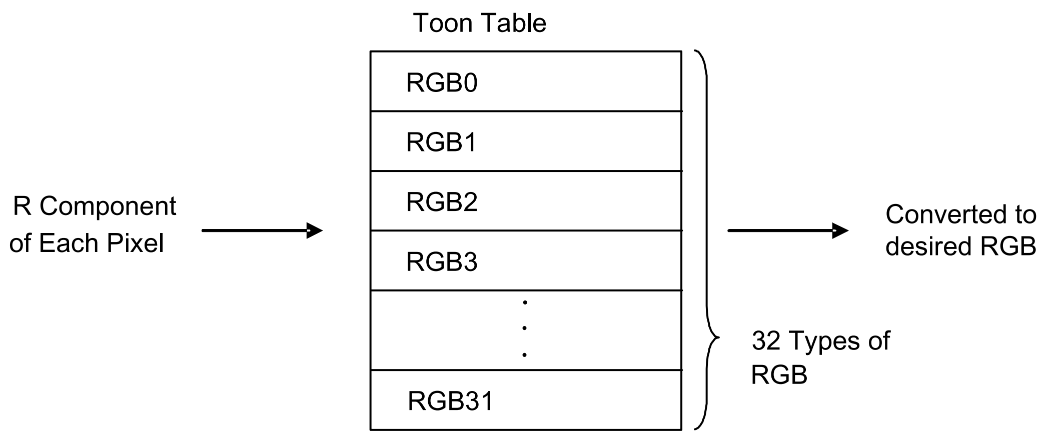
TWL-06-0156-001-A
Released: July 8, 2009

4.3.4.3 Toon Shading

Toon shading uses the toon table, which is a table used to convert RGB values.

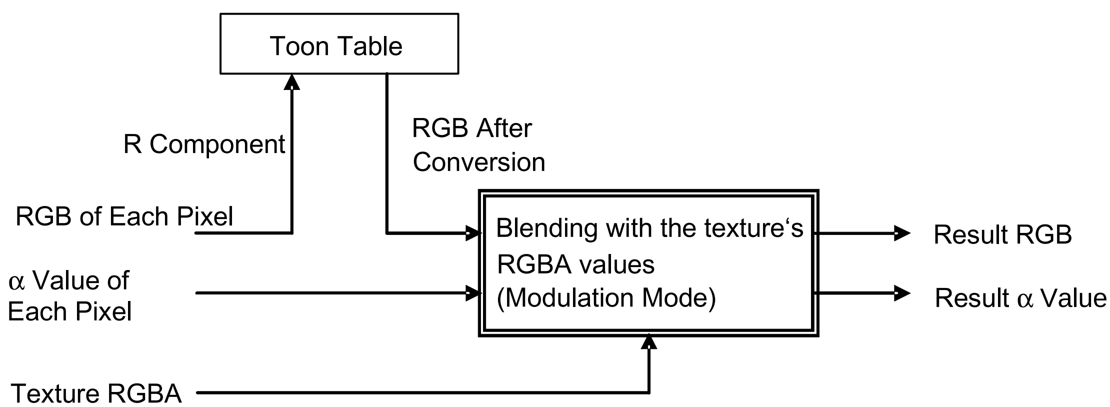
Thirty-two different colors (RGB) can be set in the toon table. Each pixel's R component is used as a lookup for RGB value conversion. (See Figure 4-25.)

Figure 4-25 Toon Table



Toon shading takes the RGB converted by the toon table and sets it as the new pixel RGB, then blends that with the texture’s RGBA values. To blend with the texture’s RGBA values, the previously mentioned Modulation Mode is used. (See Figure 4-26.)

Figure 4-26 Toon Shading

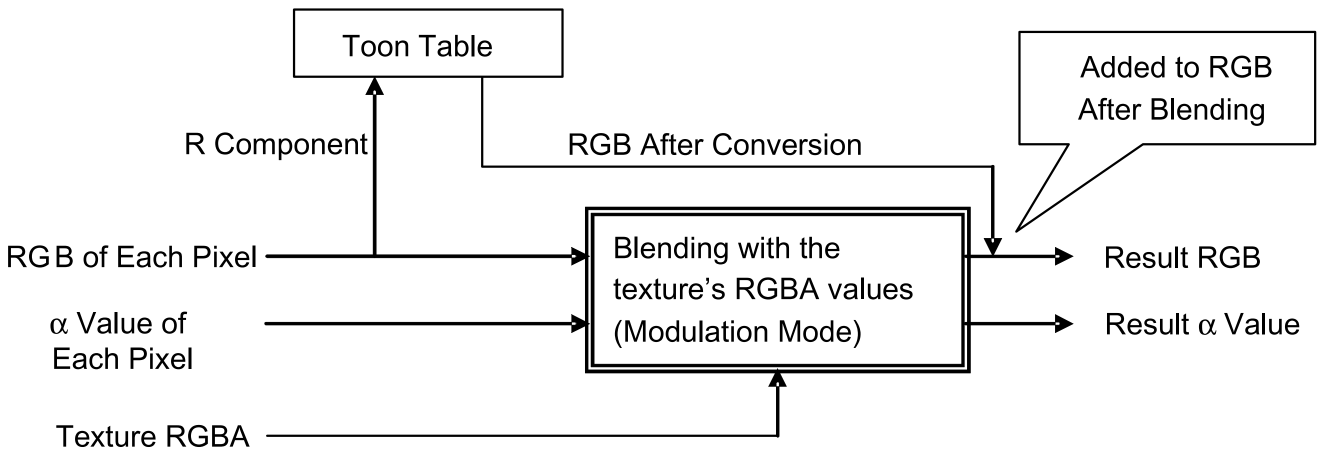


4.3.4.4 Highlight Shading

Like toon shading, highlight shading uses the toon table to convert RGB values.

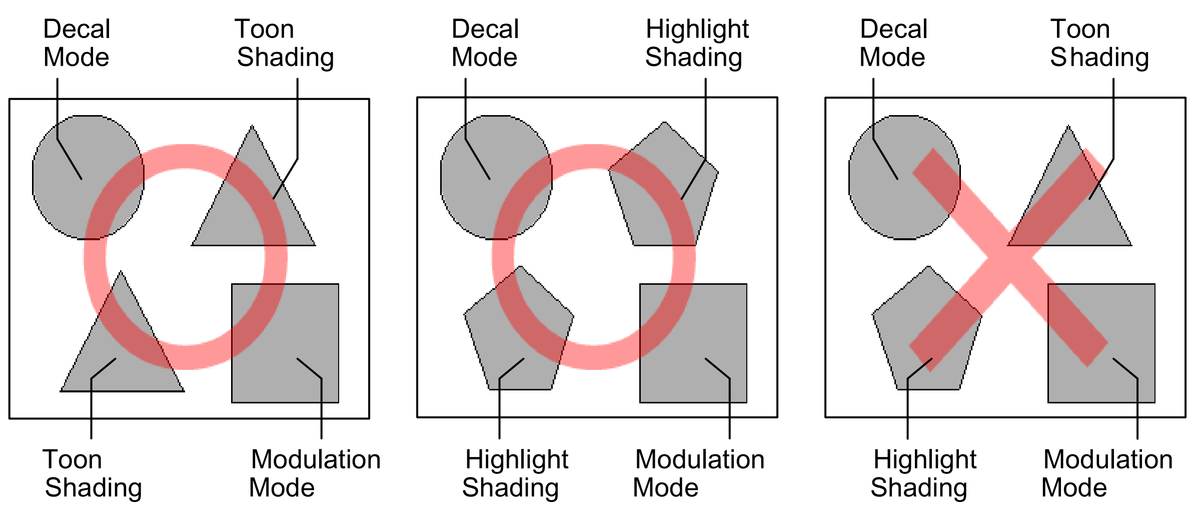
With highlight shading, each pixel's R component is first set to a new pixel and blended with the texture's RGBA values. Then, an RGB converted using the toon table is added to the blended result RGB. To blend with the texture's RGBA values, the previously mentioned Modulation Mode is used. (See Figure 4-27.)

Figure 4-27 Highlight Shading



Caution: Due to hardware specifications, you can select only either toon shading or highlight shading for each rendering frame unit. In other words, within the same rendering frame, you cannot apply toon shading to one polygon and highlight shading to another. (See Figure 4-28.)

Figure 4-28 Toon Shading and Highlight Shading Cautions

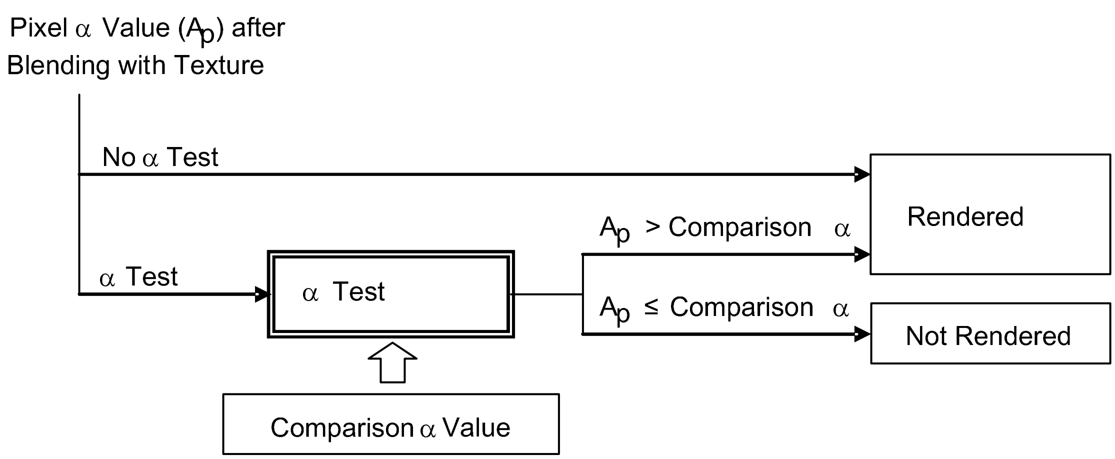


4.3.5 α Test

α test is a feature that prevents a pixel from being rendered if the pixel's α value is below a designated comparison α value.

The pixel α value after blending with the texture is used for the α test. (See Figure 4-29.)

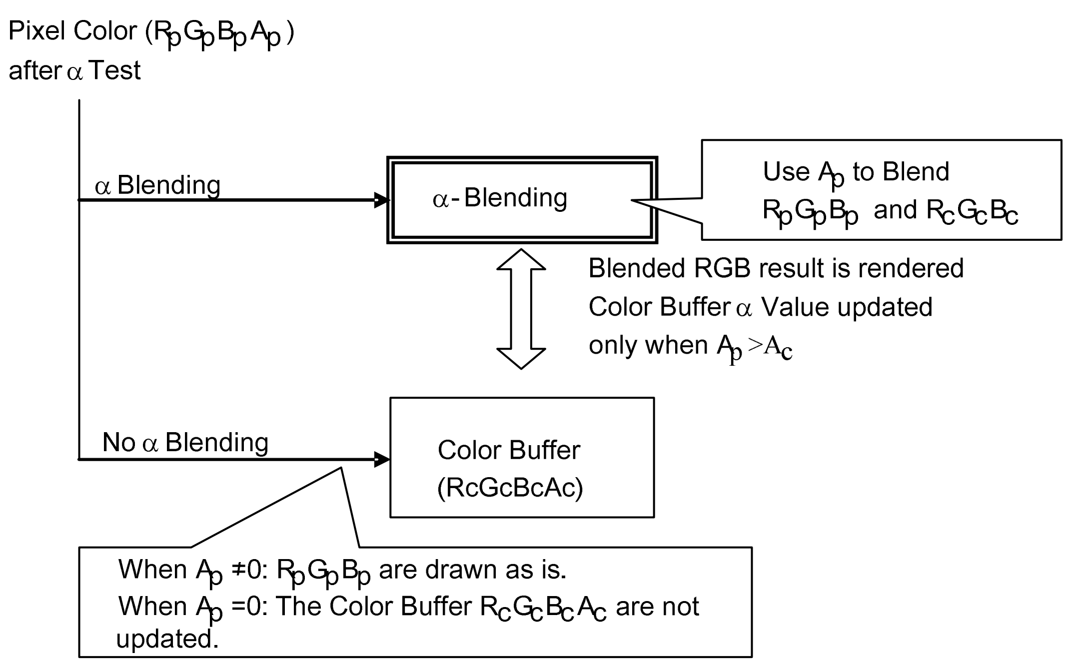
Figure 4-29 α Test



4.3.6 α Blending

α Blending is a feature that uses the pixel α value to blend the pixel RGB and color buffer RGB, and then renders the result to the color buffer. (See Figure 4-30.)

Figure 4-30 α Blending

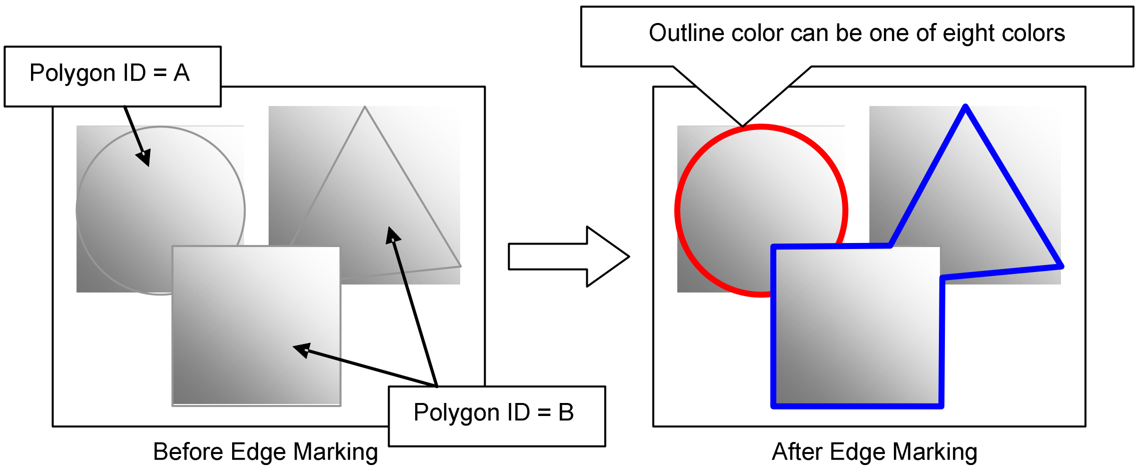


4.3.7 Edge Marking

Edge marking draws an outline around opaque polygons that have the same polygon ID.

The outline color can be selected from any eight configured colors. (See Figure 4-31.)

Figure 4-31 Edge Marking



4.3.8 Fog Blending

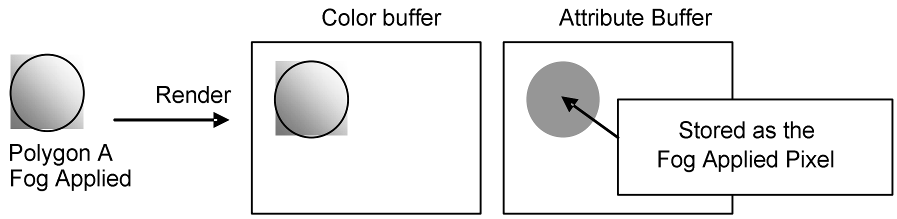
When rendering a polygon, each polygon can be specified as having or not having fog applied to it.

Polygons specified as having fog applied have a dedicated buffer, the *attribute buffer*, in which information about the polygon is stored. After all polygons have subsequently been rendered, the fog color blending process is carried out for the color buffer's corresponding pixel. (See Figure 4-32.)

Figure 4-32 Fog Blending Process Flow

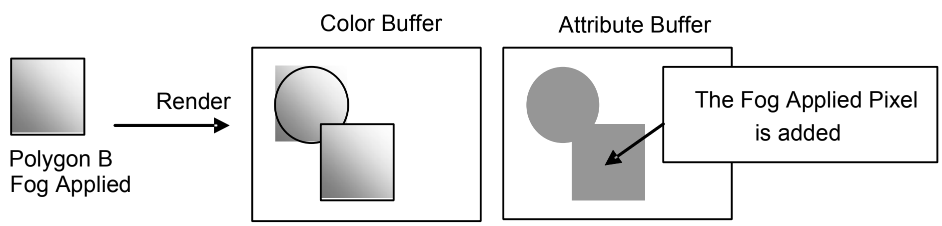
1. Rendering Polygon A with Fog Applied

The polygon A region is stored in the attribute buffer.



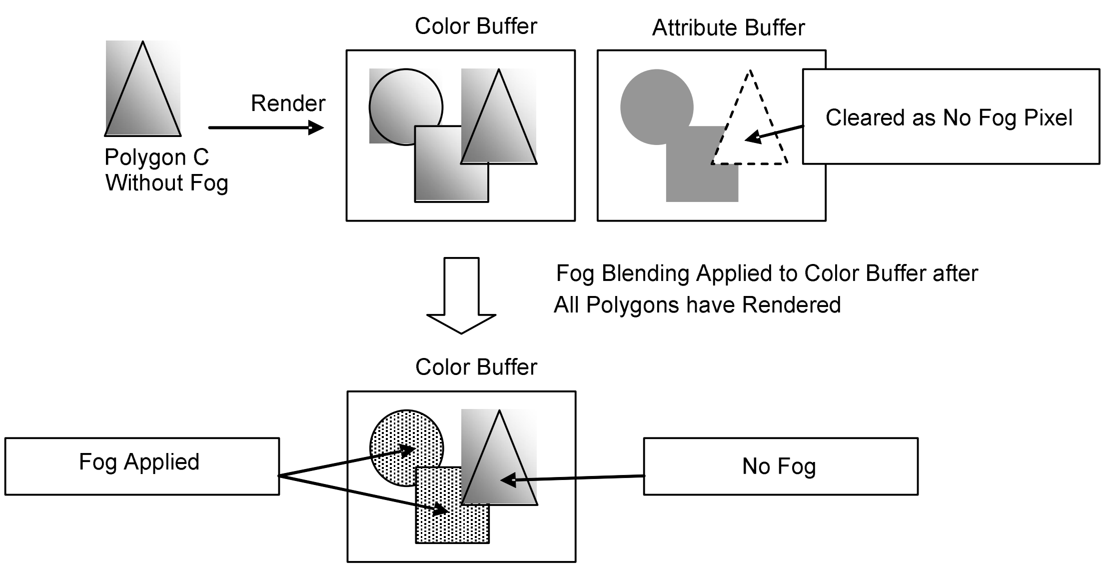
2. Rendering Polygon B with Fog Applied

The polygon B region is added to the attribute buffer.



3. Rendering Polygon C without Fog Applied

The polygon C region is cleared as the no fog pixel in the attribute buffer.



4.3.8.1 Parameters That Can Be Set for Fog

The following parameters can be used in calculating fog.

- Fog Color Alpha (RGBA)

Specifies the fog color and α.

- Fog Mode

Determines the method of fog blending. There are two methods:

  - o The fog RGBA is blended with the color buffer RGBA
  - o The color buffer RGB is kept as is, and the fog α value is mixed with the color buffer α value.

- Fog Density Table

Used to determine the fog density from the pixel depth value.

Fog application can be controlled with the fog density table.

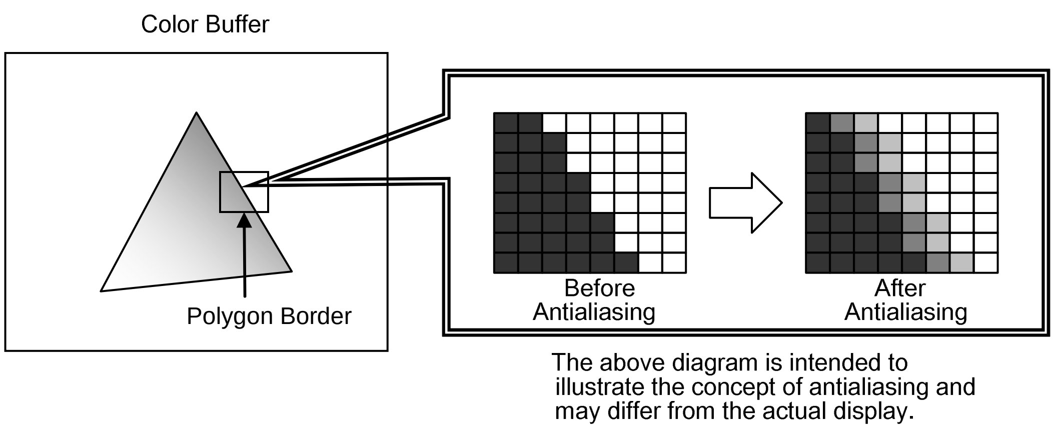
4.3.9 Antialiasing

Antialiasing is a feature used to reduce the jagged borders of rendered polygons by blending the colors of adjacent pixels (1 pixel to the top, left, right, and bottom) of the borders. (See Figure 4-33.)

When using antialiasing, the hardware determines the color blending method's calculations. Antialiasing cannot be controlled in software.

For more information on antialiasing, see the relevant chapters in the *TWL Programming Manual*.

Figure 4-33 Antialiasing



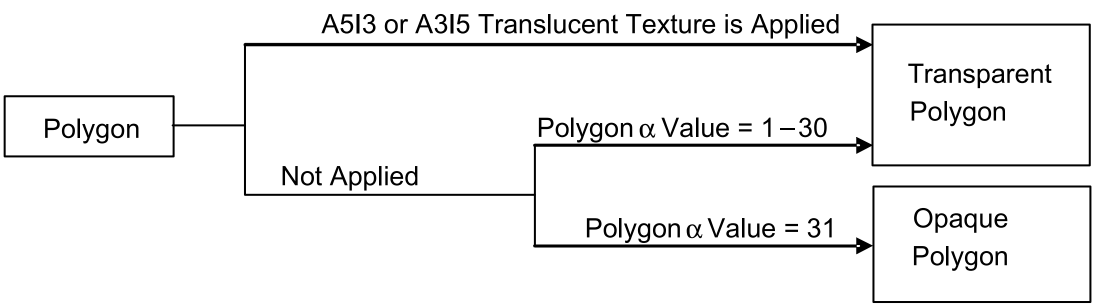
4.3.10 Supplemental 2: Rendering Sequence of Opaque Polygons and Translucent Polygons

Polygons handled by the RE are divided into opaque polygons and translucent polygons.

Polygons Handled by RE
- Opaque Polygons
- Translucent Polygons

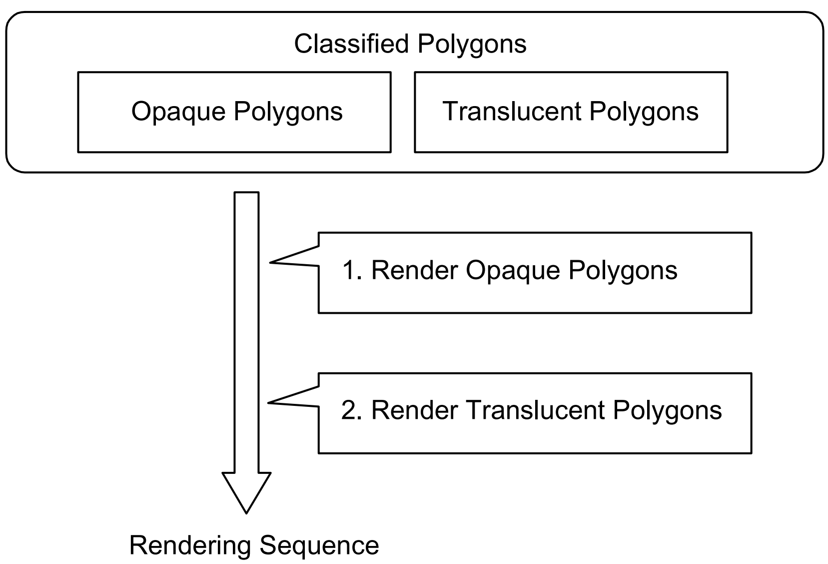
Polygons are classified as opaque or translucent according to the method shown in Figure 4-34.

Figure 4-34 Classification of Opaque and Translucent Polygons



Classified polygons will be rendered in order from opaque to translucent, as shown in Figure 4-35.

Figure 4-35 Rendering Sequence of Opaque and Translucent Polygons



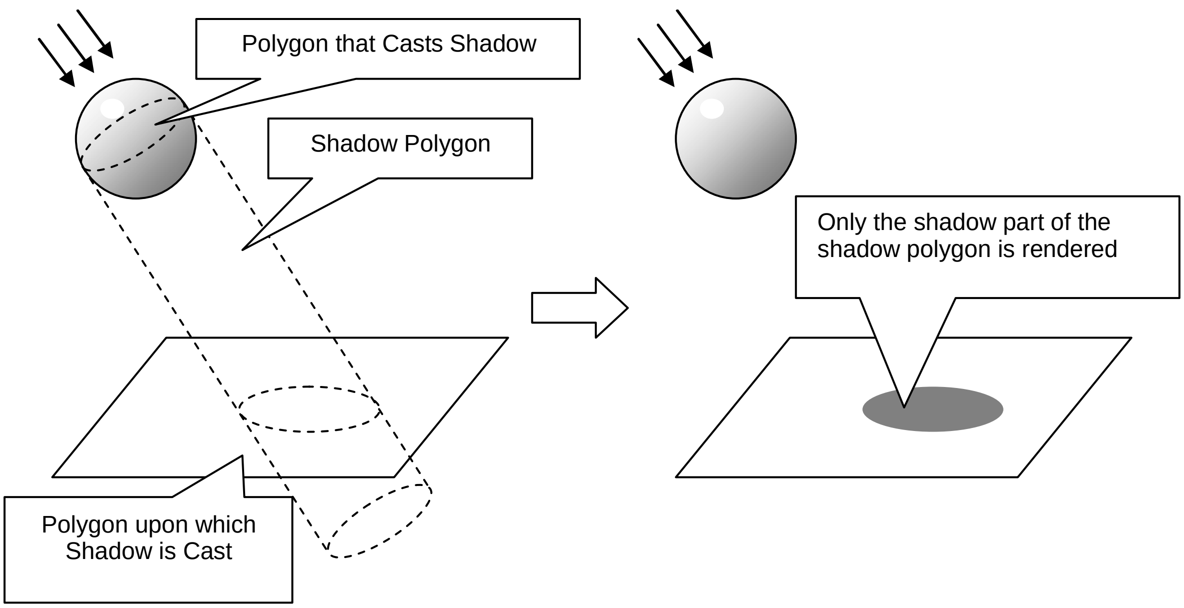
4.3.11 Supplemental 3: Shadow Polygon

Nintendo DS supports shadow rendering in hardware with a shadow volume method as a means of casting shadows on a polygon. The shadow volume method prepares the region of the polygon where light is obstructed (the spatial region that will become a shadow) as a shadow polygon, and is a technique for determining which region will become the shadow from the intersection of the shadow polygon and the ground plane. (See Figure 4-36.)

When rendering the designated shadow polygon, the RE uses the internally stored stencil buffer to create and render the shadow region shape. The stencil buffer is dedicated to shadow polygon use.

The polygon that will become the shadow volume specified by the shadow polygon is not automatically generated by the hardware. It must be generated on the software side.

Figure 4-36 Shadow Polygons



4.3.12 Supplemental 4: Initializing the Color and Depth Buffers

The color and depth buffers are initialized for each render frame.

There are two types of initialization for the color and depth buffers:

- Initialization with values

  The color and depth buffers are uniformly initialized with a color (RGBA) and a depth value.

- Initialization with image data

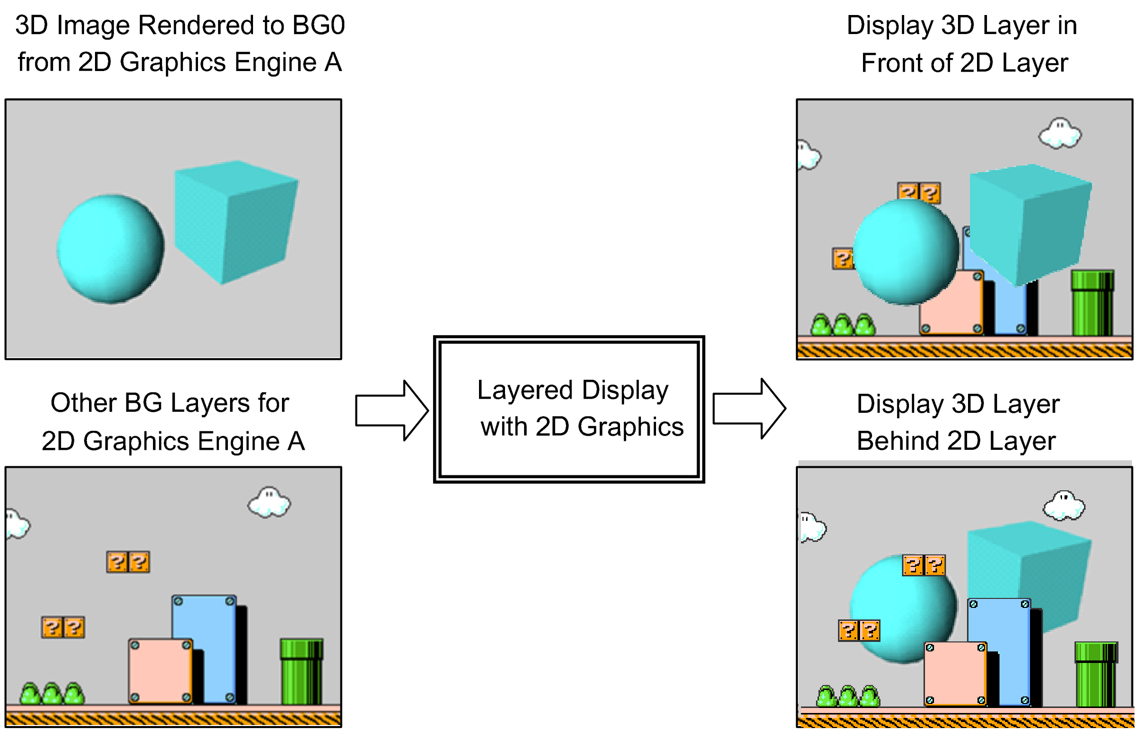
  The color and depth buffers are initialized with color image data and depth image data prepared in VRAM.

5 Merging 2D and 3D Layers

5.1 Combining 2D and 3D Layers

Images rendered with 3D graphics are rendered to the BG0 screen of 2D Graphics Engine A, according to hardware specifications. By layering the 2D Graphics Engine A's other BG layers and OBJs, the 3D graphics rendered on the BG0 layer of 2D Graphics Engine A can be combined with the 2D layers for display. (See Figure 5-1.)

Figure 5-1 Combining 2D and 3D Layers

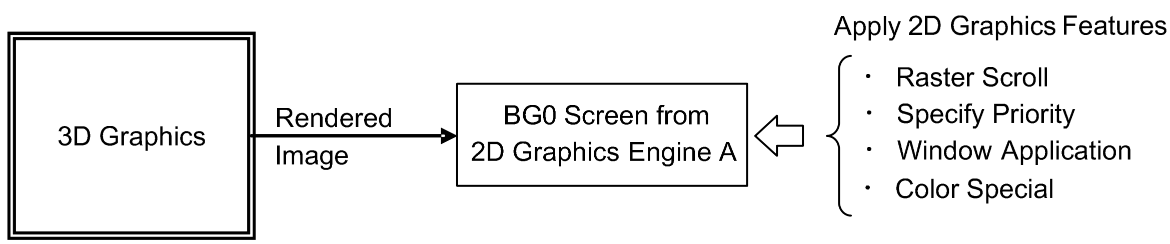


5.2 Applying 2D Graphics Features to the 3D Layer

The following 2D graphics features can be applied to the 3D graphics rendered on the BG0 layer. (See Figure 5-2.)

- Raster scroll (only horizontally)
- Specify priority with 2D Graphics Engine A's other BG screens
- Window application
- Color special effects

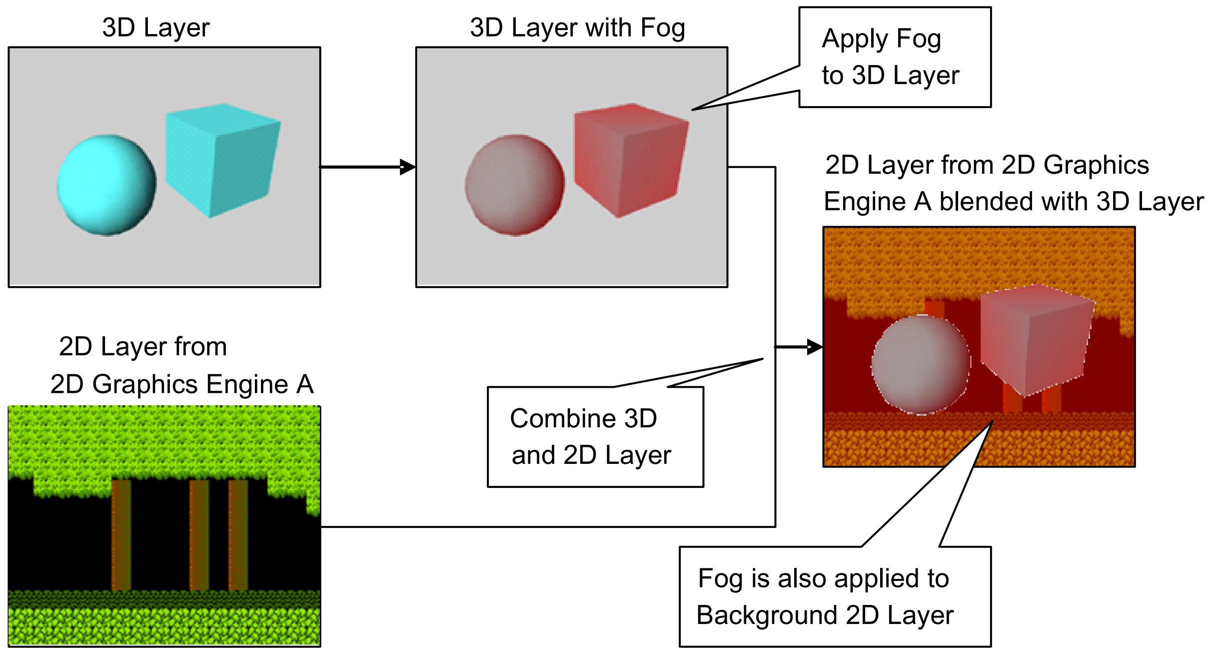
Figure 5-2 Applying 2D Graphics Features to the 3D Screen



5.3 Applying 3D Graphics Features to the 2D Layer

A 3D graphics fog can also be applied to the 2D Graphics Engine A 2D layer combined with the 3D layer. (See Figure 5-3.)

Figure 5-3 Applying 3D Graphics Features to the 2D Layer



Caution: The blending of 2D and 3D layers described in this paragraph cannot be applied to 2D Graphics Engine B.

6 Display Capture

Display capture is a feature that records image data to VRAM without displaying any of the following:

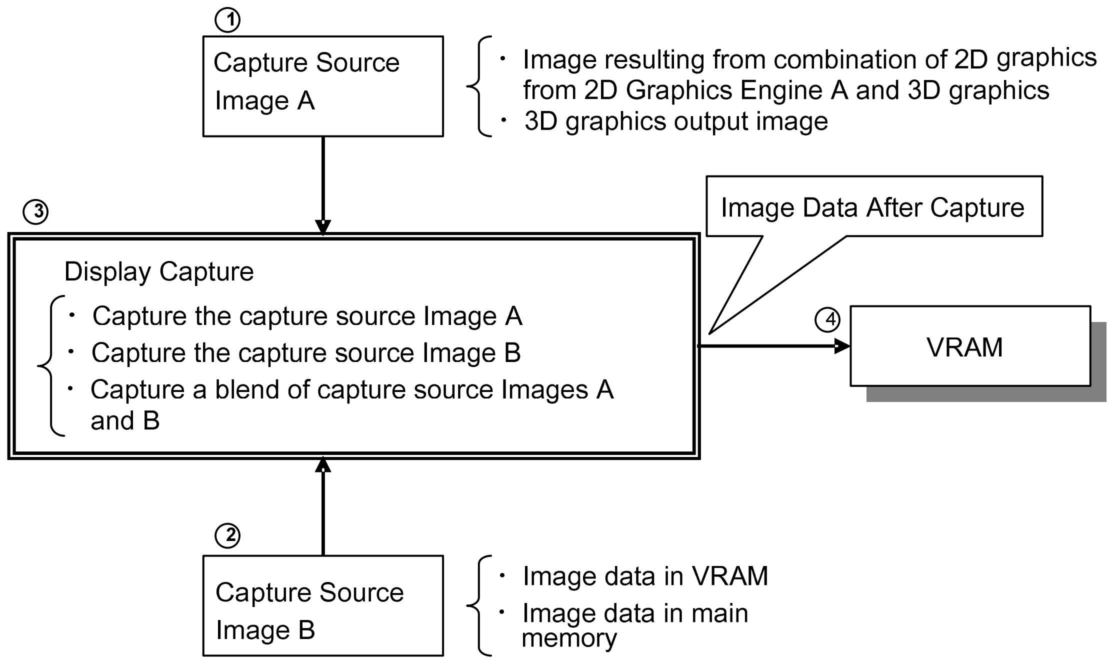
- The 2D Graphics Engine A / 3D graphics output image
- The image data in VRAM
- The image data in main memory to the LCD

This feature is carried out by the following sequence.

What it means is capture dumps the output of all those things into VRAM and does display them.

1. The capture source image A is set from one of the following two types.
   - The image output from the combination of 2D from 2D Graphics Engine A and 3D graphics
   - The 3D graphics output image
2. The capture source image B is set from one of the following two types.
   - Image data in VRAM
   - Image data in main memory
3. There are three ways of capturing the source images described in steps 1 and 2:
   - Capture the capture source A image only
   - Capture the capture source B image only
   - Capture a blending of capture sources A and B (the blending ratio can be specified)
4. The image data captured with one of the methods described in step 3 is then stored in VRAM.

Figure 6-1 Display Capture Feature

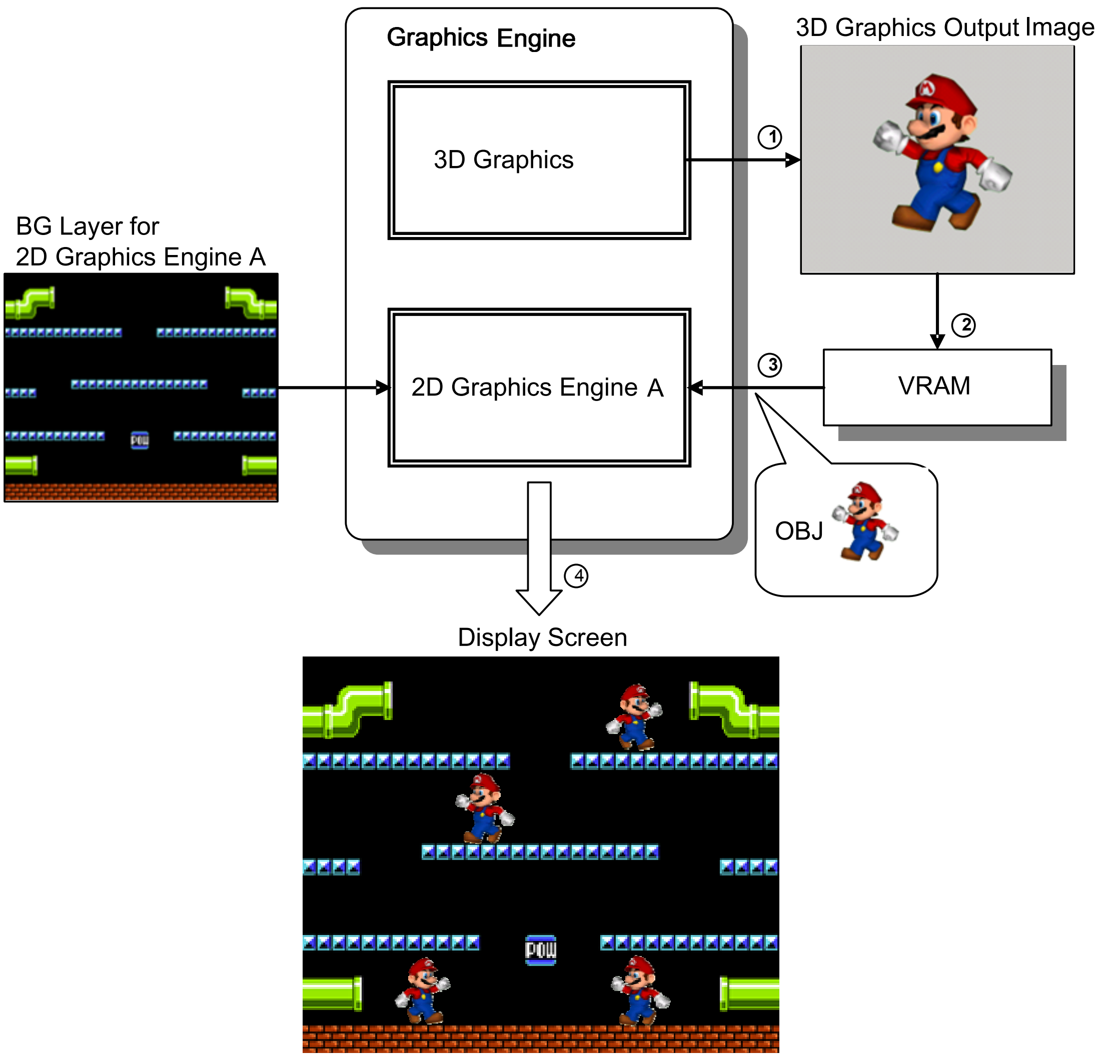


6.1 An Example of Using the Display Capture Feature

The following is an example of using the display capture feature to capture a 3D graphics output image and to use it as an OBJ for 2D Graphics Engine A. (See Figure 6-2.)

1. Render the 3D image.
2. Capture the 3D image in VRAM with the display capture feature.
3. The captured 3D image is used as an OBJ with 2D graphics **from the next render frame on.**
4. BGs and OBJs are layered and displayed by 2D Graphics Engine A.

Figure 6-2 An Example of Using the Display Capture Feature



Caution: The display capture features described in this section cannot be applied to 2D Graphics Engine B.

7 Appendix

This appendix uses each graphics feature described in the previous sections of this manual to give examples of what can be produced on the hardware overall.

7.1 Using Graphics Display Mode—Example 1

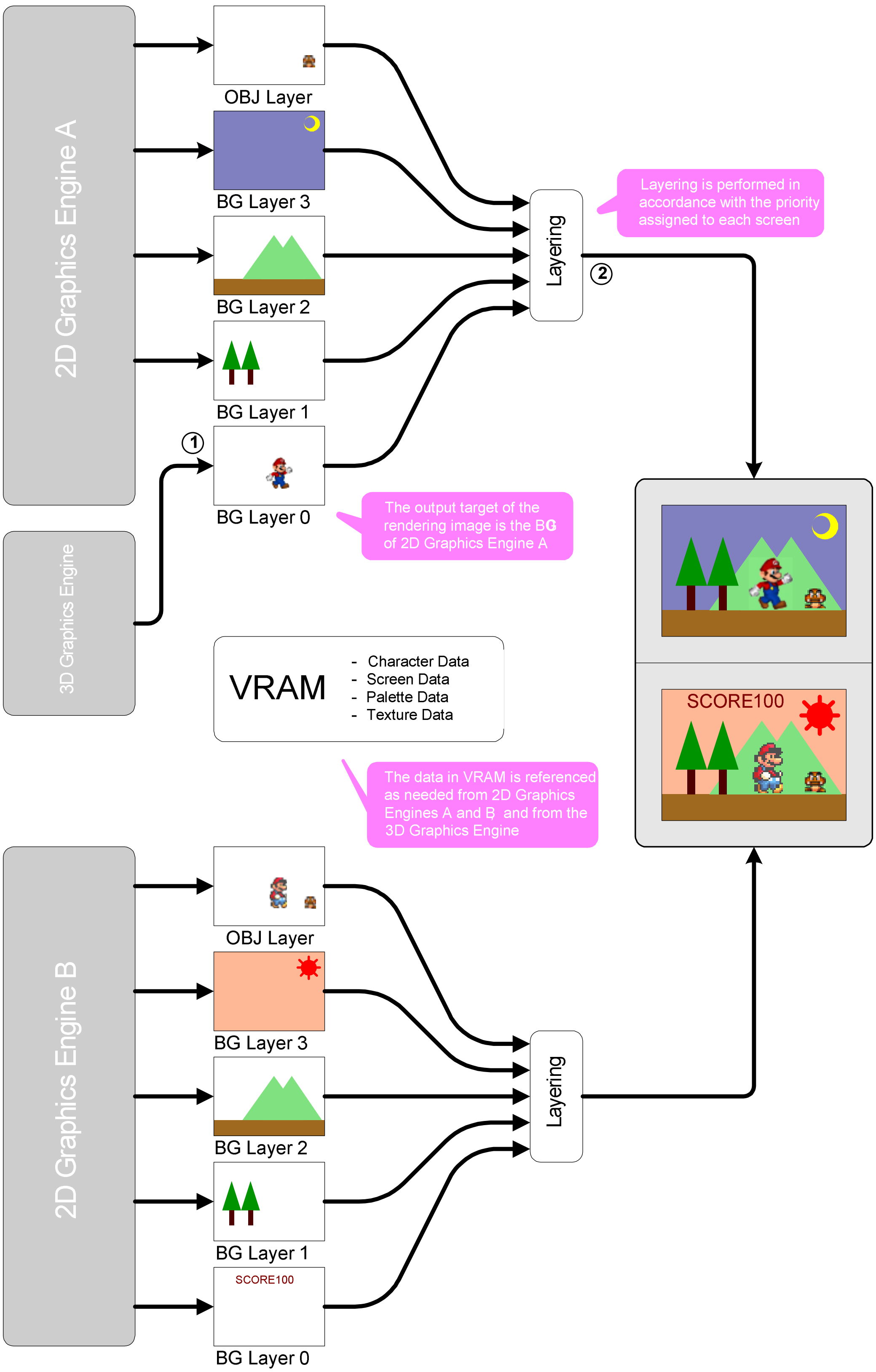
Figure 7-1 is an example of using the Graphics Display Mode to display layered 2D and 3D images. Images generated via the 3D Graphics Engine are rendered to BG0 layer of 2D Graphics Engine A, and can be displayed as layers of other BG layers and OBJ layers.

Explanation of the Diagram:

1. The 3D image generated by the 3D Graphics Engine is rendered to BG0 layer of 2D Graphics Engine A.
2. BG0, where the 3D image has been rendered is layered with other BG layers and OBJ layers, then displayed to the LCD.

TWL-06-0156-001-A
Released: July 8, 2009

Figure 7-1 Using Graphics Display Mode—Example 1



7.2 Using Graphics Display Mode—Example 2 (The Capture Feature)

Figure 7-2 is an example of the use of the Capture Feature. The 3D image generated by the 3D Graphics Engine is taken into VRAM via the Capture Feature, then used as a bitmap OBJ.

Explanation of the Diagram:

1. The 3D image is generated by the 3D Graphics Engine.
2. The generated 3D image is taken into VRAM via the Capture Feature.
3. The image data is taken into VRAM and used as a bitmap OBJ starting with the next and subsequent display frames.

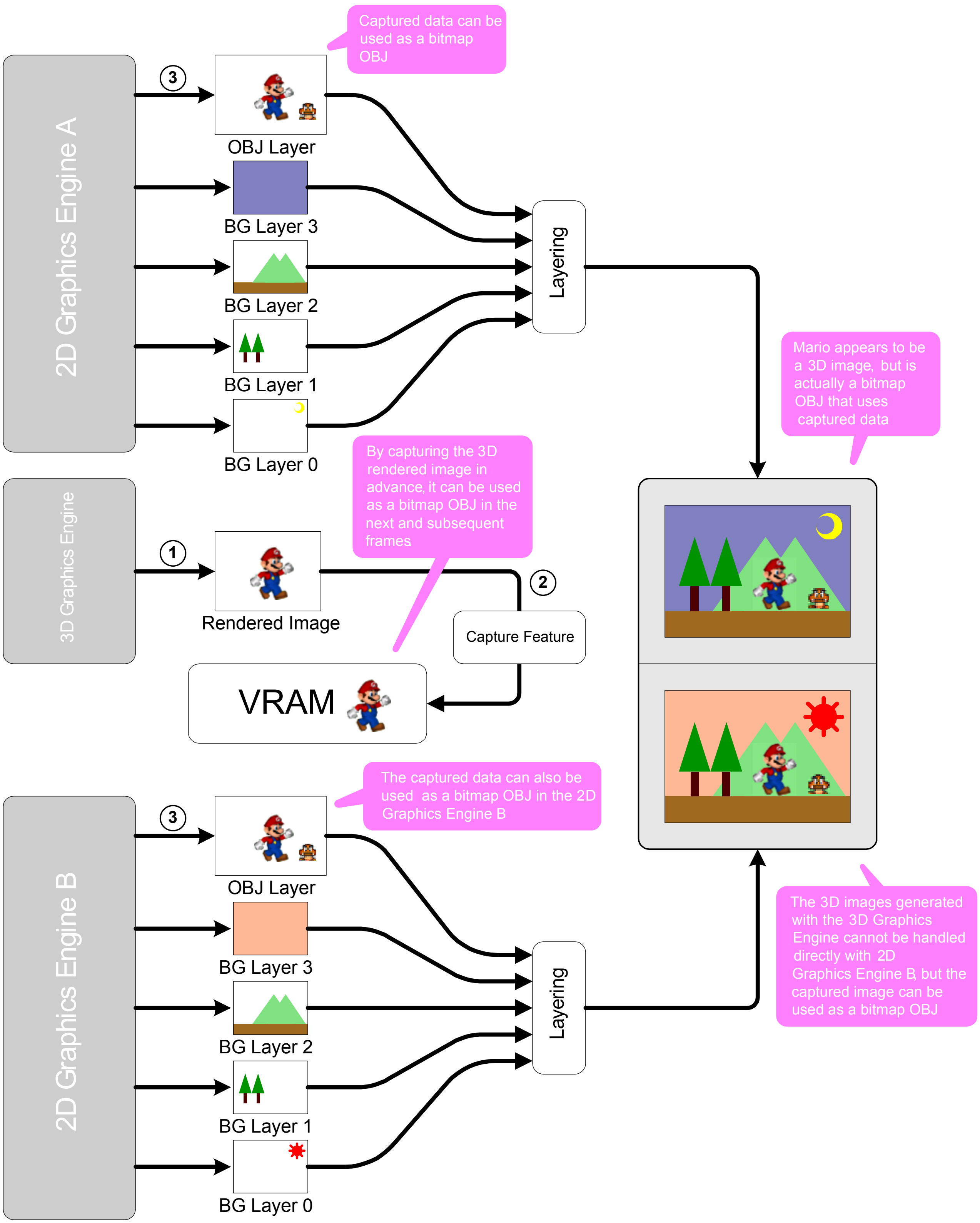
A Note for Programmers

The VRAM that records the captured data and the VRAM used for displaying the OBJ data must be prepared separately. Using two VRAMs, one for capture and one for OBJ display, you will need some sort of method for switching the VRAM roles in each frame.

Also, 2D Graphics Engines A and B must each have separate VRAM for OBJ display. Therefore, if you want to use the captured image in the VRAM allocated to 2D Graphics Engine A at the same time as 2D Graphics Engine B, you must copy the image to the VRAM allocated to 2D Graphics Engine B.

For example, you used the captured image as a bitmap OBJ, but you can also use it as a bitmap BG.

Figure 7-2 Using Graphics Display Mode—Example 2 (The Capture Feature)



7.3 Using Graphics Display Mode—Example 3 (Displaying 3D on Two Screens)

Because there is only one 3D Graphics Engine built into Nintendo DS, 3D images cannot be displayed simultaneously on both the Main LCD and the Sub LCD. However, by changing the output target of the 3D Graphics Engine in each frame (which will cause a refresh every other frame), 3D images can be displayed on both LCDs.

As an example, a normal Mario is on the Main LCD, and Luigi is on the Sub LCD. Figure 7-3 shows the process in the odd frames, while Figure 7-4 shows the process in the even frames.

Explanation of the Diagram (Odd Frames)

1. Mario is rendered in the 3D Graphics Engine.
2. To prepare the next display frame, capture the image consisting of the layered 2D and 3D layers into VRAM-D.
3. Switching the main LCD output from the VRAM Display Mode (VRAM-D) of 2D Graphics Engine A makes it possible to display image data from two frames prior to the current one saved in VRAM-D simultaneous to performing capture.
4. The image in VRAM-C captured in the previous display frame is displayed as a bitmap BG on the Sub LCD.

Explanation of the Diagram (Even Frames)

1. Luigi is rendered in the 3D Graphics Engine.
2. To prepare the next display frame, capture the image consisting of the layered 2D and 3D layers into VRAM-C.
3. Switching the sub LCD output from the VRAM Display Mode (VRAM-C) of 2D Graphics Engine A makes it possible to display image data from two frames prior to the current one saved in VRAM-C simultaneous to performing capture.
4. The image in VRAM-D captured in the previous display frame is displayed as a bitmap OBJ on the Main LCD.

A Note for Programmers

There is a reason for the display of the image in VRAM-C as a BG for the odd frames, and the display of the image in VRAM-D as an OBJ for the even frames.

One of the specifications of Nintendo DS is that when allocating VRAM-C in 2D Graphics Engine B, its application is limited to BG data reference. In the same way, when allocating VRAM-D to 2D Graphics Engine B, its application is limited to OBJ data reference. It is due to these specifications that BG and OBJ are being used separately.

This algorithm leverages the fact that pre-capture VRAM content is rendered when the VRAM Display Mode and the capture destination both specify the same VRAM.

Figure 7-3 An Example of Displaying 3D on Both the Main and Sub LCDs (Odd Frames)

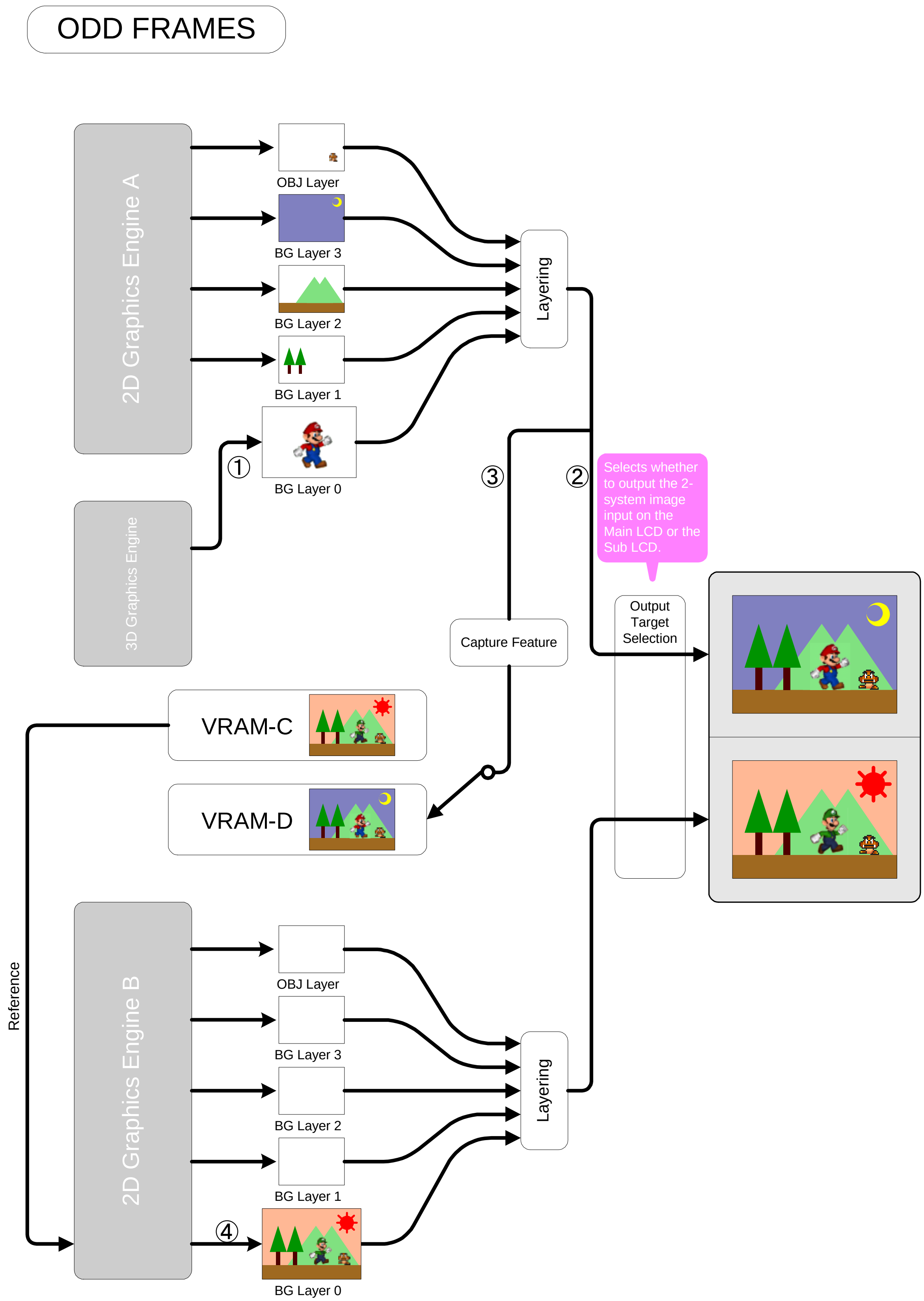
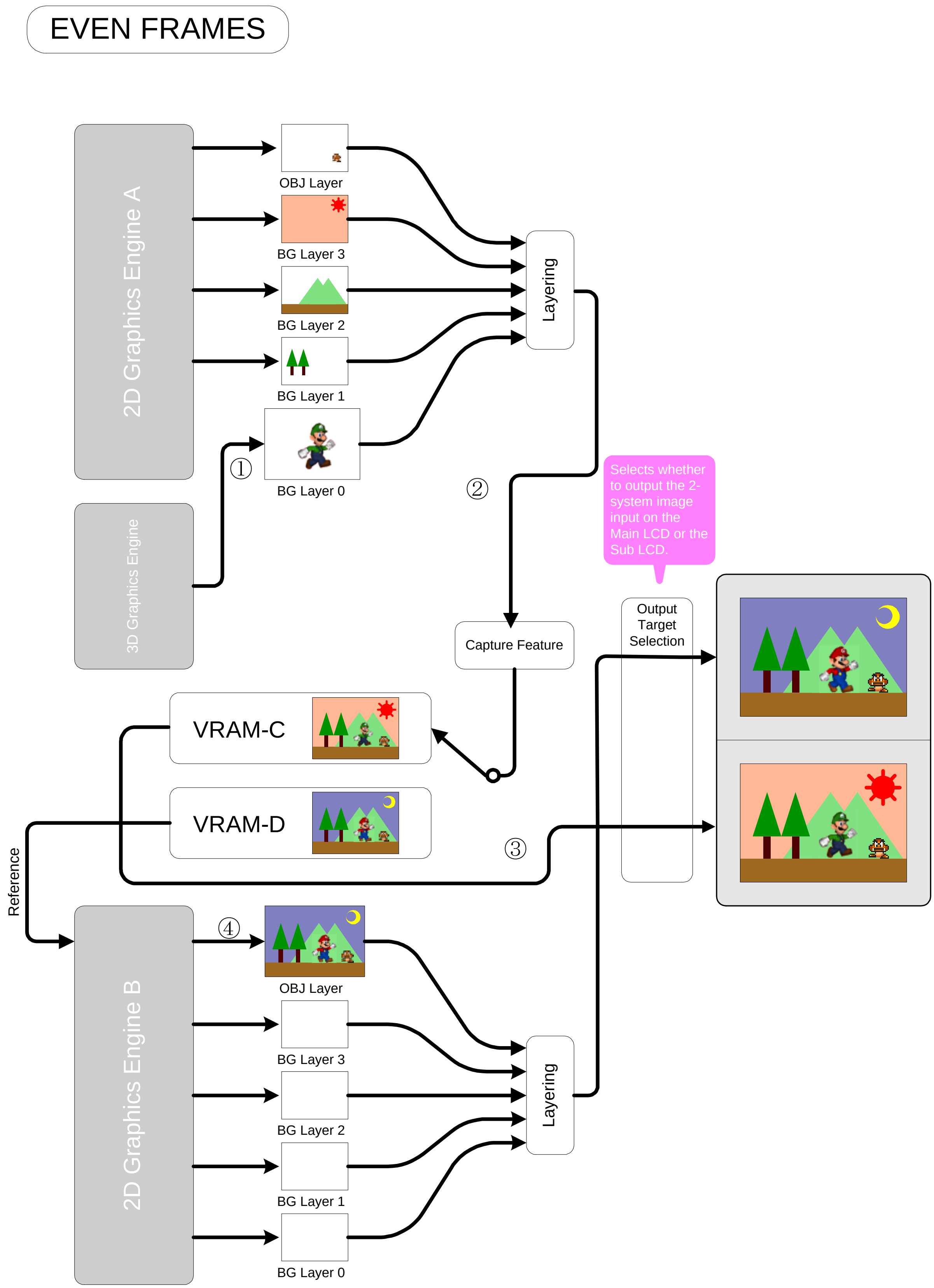


Figure 7-4 An Example of Displaying 3D on Both the Main and Sub LCDs (Even Frames)



7.4 An Example of Using VRAM Display Mode

In VRAM Display Mode, one frame's worth of bitmap data in VRAM can be displayed. In the example shown in Figure 7-5, the image generated by the graphics circuit and the image being currently displayed are blended. The result is taken into VRAM using the capture feature, allowing for a Motion Blur Effect.

Explanation of the Diagram:

1. The bitmap data in VRAM is displayed on the LCD.
2. The image displayed on the LCD is set to capture source B.
3. The image generated by the graphics circuit is newly set to capture source A.
4. The capture source A and B images are blended.
5. After blending, the image is taken into VRAM with the capture feature. The image is then displayed in the next frame.

A Note for Programmers

In VRAM Display Mode, the captured image can be written to the VRAM used for display.

Figure 7-5 An Example of Using VRAM Display Mode (Motion Blur Effect)